VI Brazilian Symposium on Computer Games and Digital Entertainment

November, 7-9, 2007
São Leopoldo – RS - BRAZIL

PROCEEDINGS

Published by
Sociedade Brasileira de Computação - SBC

Edited by

Marcelo Walter
Luiz Gonzaga da Silveira Jr

Computing Track Chairs
Marcelo Walter
Bruno Feijó
Jorge Barbosa

Organized by

Soraia Raupp Musse
Fernando Santos Osório
João Ricardo Bittencourt
Luiz Gonzaga da Silveira Jr
Christian Hofsetz

Universidade do Vale do Rio dos Sinos - UNISINOS
Pontifícia Universidade Católica do Rio Grande do Sul - PUCRS

Sponsored by

Sociedade Brasileira de Computação

Table of Contents

SBGames 2007

PREFACE

Welcome to **SBGames 2007**, the VI edition of the Brazilian Symposium on Computer Games and Digital Entertainment. SBGames is the yearly symposium of the Special Interest Group on Games and Digital Entertainment of the Brazilian Computer Society (SBC).
This volume contains the 24 full papers accepted for the computing track, out of 55 submitted, an acceptance ratio of 44%. Out of the 24 accepted papers, 13 are in English (54%). We hope this trend will continue, increasing the visibility of the research work being developed in Brazil by the gaming community. For the first time the selection process was double blind, and each paper was reviewed by at least 3 experts, improving the quality of the reviewing process. Also, a selection of the best papers will be selected for publication in a special edition of IJCGT - International Journal of Computer Games Technology.

Papers accepted for the art & design, game & culture tracks, short papers accepted for all tracks and screenshot of games selected to the game festival have been included as addition material.

We would like to thank all authors, whose work and dedication made possible to put together an exciting program. Next, we would like to thank all members of the technical program committee and reviewers, for their time helping us maintain the overall quality of the program.

We would like to wish all attendees an exciting symposium!

São Leopoldo, November 2007

Marcelo Walter
Bruno Feijó
Jorge Barbosa

Chairs of the program committee
Computing track

Program Committee

Adelailson Peixoto	Universidade Federal de Alagoas
Alexandre Sztajnberg	Universidade do Estado do Rio de Janeiro
André Campos	Universidade Federal do Rio Grande do Norte
Bruno Feijó	Pontifícia Universidade Católica do Rio de Janeiro
Cesar Pozzer	Universidade Federal de Santa Maria
Christian Hofsetz	Universidade do Vale do Rio dos Sinos
Clauirton Siebra	Universidade Federal de Pernambuco
Drew Davidson	Carnegie Mellon University
Edmond Prakash	Manchester Metropolitan University
Edson Cáceres	Universidade Federal do Mato Grosso do Sul
Esteban Clua	Universidade Federal Fluminense
Fernando Osório	Universidade do Vale do Rio dos Sinos
Fernando Trinta	Universidade Federal de Pernambuco
Flávio S. C. da Silva	Universidade de São Paulo
Geber Ramalho	Universidade Federal de Pernambuco
Jacques Brancher	Universidade Regional Integrada - Campus de Erechim
Jim Terkeurst	University of Teesside
João Comba	Universidade Federal do Rio Grande do Sul
John Buchanan	Carnegie Mellon University
Jorge Barbosa	Universidade do Vale do Rio dos Sinos
José Saito	Universidade Federal de São Carlos
Judith Kelner	Universidade Federal de Pernambuco
Laércio Ferracioli	Universidade Federal do Espírito Santo
Luiz Chaimowicz	Universidade Federal de Minas Gerais
Manuel M. Oliveira Neto	Universidade Federal do Rio Grande do Sul
Marcelo Dreux	Pontifícia Universidade Católica do Rio de Janeiro
Marcelo Walter	Universidade Federal de Pernambuco
Maria Andréia Rodrigues	Universidade de Fortaleza
Martin Hanneghan	Liverpool John Moores University
Michael Youngblood	The University of North Carolina at Charlotte
Patrícia Tedesco	Universidade Federal de Pernambuco
Paulo Pagliosa	Universidade Federal de Mato Grosso do Sul
Paulo Rodacki Gomes	FURB - Universidade Regional de Blumenau
Romero Tori	SENAC-SP / Universidade de Sao Paulo
Sérgio Scheer	Universidade Federal do Paraná
Soraia Musse	Pontifícia Universidade Católica do Rio Grande do Sul
Waldemar Celes	Pontifícia Universidade Católica do Rio de Janeiro
Wu Shin-Ting	Universidade Estadual de Campinas

Reviewers

Adelailson Peixoto
Alex Gomes
Alexandre Sztajnberg
André Campos
Bruno Feijó
Carlos Dietrich
Cesar Pozzer
Christian Hofsetz
Christian Pagot
Clauirton Siebra
Danielle Rousy Silva
Denison Tavares
Drew Davidson
Edmond Prakash
Edson Cáceres
Eduardo Jacober
Eleri Cardozo
Esteban Clua
Fernando Osório
Fernando Trebien
Fernando Trinta
Flávio S. C. da Silva
Geber Ramalho
Harlen Batagelo
Jacques Brancher
Jim Terkeurst
Joao Bittencourt
João Bernardes
John Buchanan
Jorge Barbosa
José Saito
Judith Kelner
Laércio Ferracioli
Leandro Fernandes
Leonardo Schmitz
Luiz Chaimowicz
Luiz H. de Figueiredo
Marcelo de P. Guimarães
Marcelo Dreux
Marcelo Walter
Marcos Slomp
Maria A. Rodrigues
Marinho Barcellos
Martin Hanneghan
Mauro Steigleder
Michael Youngblood
Patrícia Tedesco
Paulo Pagliosa
Paulo Rodacki Gomes
Rafael Torchelsen
Ricardo Nakamura
Romero Tori
Samir Souza
Sérgio Scheer
Soraia Musse
Waldemar Celes
Wu Shin-Ting

Technical Papers

Creating a Director for an Interactive Storytelling System

Vinicius da Costa de Azevedo Cesar Tadeu Pozzer

Universidade Federal de Santa Maria – UFSM
Departamento de Eletrônica e Computação – DELC

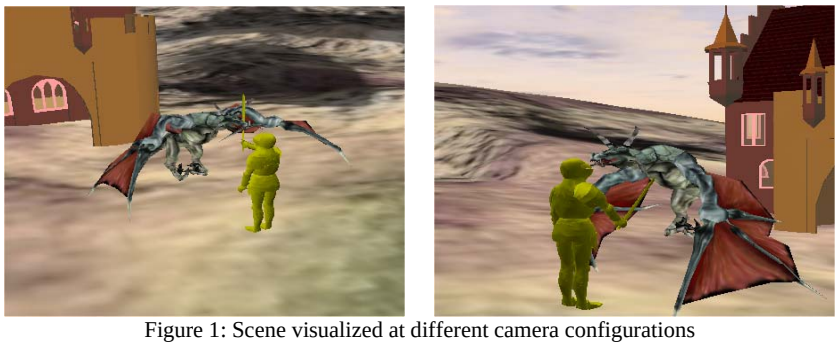


Figure 1: Scene visualized at different camera configurations

Abstract

Capturing the essence of scenes during the graphical dramatization of events in a storytelling environment is so important than generating a good and interesting plot. In this paper we propose an architecture that allows user supply parameters to enrich a plot and that, in the same time, give tips of how the camera should behave during dramatization. We propose the creation of a director, an intelligent agent that encapsulates cinematography expertise. With the inclusion of this virtual director, both actors and camera lose part of their autonomy in order to follow not more high level actions, but detailed instructions that are originated from user intervention plus cinematographic expertise like idioms and camera settings.

Keywords: Camera techniques, agents, storytelling

Authors' contact:
{azevedo, pozzer}@inf.ufsm.br

1. Introduction

Interactive Storytelling is a new medium of Digital entertainment where authors, audience, and virtual agents engage in a collaborative experience. It can be seen as a convergence of games and filmmaking [Scientific American 2000].

Storytelling systems can deal with both story generation, user interaction and dramatization. Different approaches have been proposed, using techniques and concepts from many areas such as Computer Graphics, Artificial Intelligence, Cognitive Science, Literature and Psychology. The suitability of each approach depends on the goal of each application. In this paper we concentrate our attention to the dramatization process, responsible for the graphical representation of live actors performing sequences of predefined events. More specifically, we propose techniques for specifications of an autonomous camera agent that assumes the role of a director.

Although this paper focus specially on the dramatization, it is very important to comprehend the different approaches around story models because those concepts are tightly related to the dramatization process. The story can be focused on characters or on the plot.

In a *character-based* approach [Cavazza 2002; Mateas 2000; Young 2000] the storyline usually results from the real-time interaction among virtual autonomous agents that usually incorporates a deliberative behavior. The main advantage of a character-based model is the ability of anytime user intervention, which means that the user may interfere with the ongoing action of any character in the story, thereby altering the plot as it unfolds. As a result of such strong intervention, there is no way to estimate what decisions or actions will be made by the virtual actors in order to allow the director to have same authority as occurs in a real filmmaking.

By contrast, in *plot-based* models [Spierling et al. 2002; Grasbon 2001], characters that usually incorporate a reactive behavior should follow rigid rules specified by a plot. The plot is usually built in a stage that comes before dramatization. In a pure plot-based approach, user intervention might be more limited. User intervention is not allowed during dramatization. Such approach ensures that actors may follow a predefined script of actions that are known beforehand. Such script may be built automatically from plot or with the help of the author.

The director architecture proposed in this paper was implemented over an interactive storytelling system

called LOGTELL [Ciarlini et al. 2005], which was implemented in three main modules: the IPG (Interactive plot generator) [Ciarlini et al. 2000], Plot manager and Drama manager, as shown in Figure 2. The general architecture can be seen as a pipeline where data (events) is transformed from morphological functions into real-time 3D animations dramatized by virtual actors (Figure 1) handled by a graphical engine.

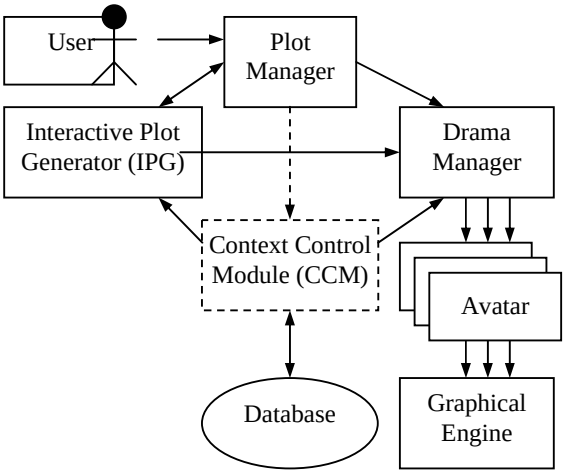


Figure 2: LOGTELL architecture (extracted from [Ciarlini et al. 2005]

In the original architecture of LOGTELL, user intervention was limited to control plot generation through the graphical interfaces supplied by Plot Manager. Dramatization process, implemented by the Drama Manager, is an automatic component that encapsulates autonomous agents represented by 3D virtual actors that perform sequences of events previously generated by IPG.

To allow full control in the way events are dramatized and recorded, in this paper we propose the creation of a director, which uses cinematographic knowledge to employ the fundamentals of filmmaking. It delegates actions to the actors and to the environment, using predefined camera settings. We also provide a script for specifying how events should be graphically dramatized. With this approach, the director acts as an intelligent agent, not just as a reactive camera; it interprets the plot built from IPG and the Plot Manager, assigning specific actions to virtual actors as the story unfolds, as well setting camera's parameters for defining takes at each moment.

The next section describes related work in the area of camera systems. Section 3 presents concepts and theory about cinematography. On section 4 we present the overall architecture of the LOGTELL and highlights new concepts about the dramatization process. Section 5 details our architecture, the camera model and idiom implementation. On section 6 we present the user interaction through the script edition system. Section 7 describes the details of the process of dramatization. Section 8 concludes our work, and presents our future researches.

2. Related Work

Much work has already been done in camera systems. There is a clear distinction between researches around character-based and plot-based applications. Plot-based applications gives access to all the actions before camera planning, allowing the system to have a greater control of the scenes based upon pure cinematography knowledge; the character-based applications do not give immersive control and planning over the story, since all the information is send in real-time to the camera system.

The basic principles of camera positioning employing cinematography knowledge in form of idioms specifications, representing the same level of abstraction as expert directors was first explored by Christianson et Al. [1996]. Defining the concept of idiom, which is widely used in researches involving camera systems, they implemented a directive camera control language.

In character-based applications, the first camera system was developed by He at al. [1996] – idioms were implemented as a set of small hierarchically organized finite state machines, the same approach we adopted in this paper. Halper et al. [2001] has proposed a camera control based upon constraint specifications; however high constraint satisfaction implies in poor frame coherence – solving this problem is the main focus of their work. Interesting algorithms for visibility and automatic camera placements were also proposed and discussed in this paper.

In plot-based applications, Courty et al. [2003] introduces a scheme for integrating storytelling and camera systems. Charles et al. [2002] have explored architectural and organizational concepts to achieve satisfactory camera planning when we have different context timeline stories that can be alternated with the flow of the time. Barros [2004] relates an interesting storytelling system, which is concerned with user interaction, but does not focus on the dramatization and more specifically on the camera control.

These approaches only reach superficial implementation issues, and do not provide an integrated interactivity with the user. Our paper proposes an interactive intelligent system, based upon high level user's specifications allied to an intelligent agent capable to employ cinematographic knowledge.

3. Cinematography Concepts

The American Society of Cinematographers defines cinematography as:

> *"a creative and interpretive process that culminates in the authorship of an original work of art rather than the simple recording of a physical event."*

This means that cinematography is a far complex process, and since we don't have the technology to develop creative and intelligent directors that approaches the human interpretation of reality, we based our work upon human specifications and some heuristics [Arijon 1976] defined by cinematographers:

- **Create a line of interest:** Is the line that connects the major two points in one scene (mostly of the times, the two actors that interact in the scene).
- **Parallel editing:** Scenes should alternate between different contexts, locations and times.
- **Only show peak moments of the story:** Repetitive movements should be eliminated.
- **Don't cross the line**: Once a scene is taken by a side of the interest line, the camera must maintain in that side, to not make confused movement shots. The camera can switch sides, but only upon an establishing shot, that shows that transition.
- **Let the actor lead:** The actor should initiate all movement, and the camera should come to rest a little before the actor.
- **Break movement:** A scene illustrating some movement must be broke into at least two shots.

Directors have also specified various camera placements relative to the line of the interest. Figure 3 illustrates possible camera placements that are used in this paper. These heuristics and camera placements have been abstracted into constraints that are used along the paper to help in the dramatization process.

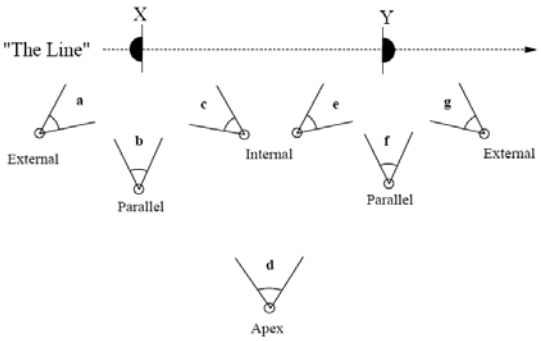


Figure 3: Camera placements relative to the line of interest (extracted from [Christianson 1996])

4. LOGTELL and Dramatization Control

LOGTELL [Ciarlini et al. 2005] is a system that encapsulates story generation and visualization. Story generation is based on the logic specification of a model of the chosen story genre, where possible actions and goals of the characters are described. The model is composed by typical events and goal-inference rules. In our experiments, stories are based on a Swords-and-Dragons context, where heroes, victims, and villains interact in a 3D scenario occupied by castles and churches. The stories are told from a third-person viewpoint.

The possible events were modeled by just a few parameterized operations, which can nevertheless generate a considerable variety of different plots. The specified operations were the following:

- go(CH,PL): character CH goes to place PL;
- reduce_protection(PL): the protection of place PL (represented by the number of guardians) is spontaneously reduced;
- kidnap(VILLAIN,VICTIM): character VILLAIN kidnaps character VICTIM;
- attack(CH,PL): character CH attacks place PL (fighting the guardians);
- fight(CH1,CH2): character CH1 fights character CH2;
- kill(CH1,CH2): character CH1 kills character CH2;
- free(HERO,VICTIM): character HERO frees character VICTIM, raising the degree of affection of VICTIM for HERO;
- marry(CH1,CH2): character CH1 marries character CH2; and
- get_stronger(CH): strength level of character CH is raised (by a magical power).

The arguments for these operations can be characters or places. In our test scenario, characters are Draco (a dragon, - the villain -, who lives in its castle), Marry (a princess, - a potential victim-, who lives in her castle), and Brian and Hoel (knights, - the heroes).

Through the plot manager interface, the user can alternate phases of plot generation, in which intervention is possible, and phases of dramatization. This intervention is always indirect. This means that the user may try to force the occurrence of desired events and situations and also accept or reject partially-generated plots. If the user does not like the partial plot, IPG can be asked to generate another alternative. If the user accepts the partial plot, the process continues by inferring new goals from the situations holding as a result of this first stage. The IPG can

generate a number of possibilities of chained events for each phase to fulfill story goals.

The plot manager is implemented in Java and comprises a number of graphical interfaces that allows user to transparently access IPG (implemented in Prolog) for control plot generation and Drama Manager for control the dramatization of the plot being building (Figure 4). We should notice that IPG generates events in a partial order, determined by the chaining of pre- and post-conditions. Into the Plot Manager, each event is represented by a rectangular box that may assume a specific color according to its current status. To determine the sequence, the user connects the events in a sequential order of his/her choice, respecting the temporal constraints supplied by IPG. To help the user in this process, we utilize colors to distinguish operations that are already connected (yellow), operations that – in view of the temporal constraints – can be immediately connected (green) or cannot yet be connected (red). The starting root is blue and the current operation being rendered is cyan. To connect two operation boxes, the user must click with the mouse over the origin and drag over the destiny (the same process is used to remove a link between two operations). Once the current plot (or part of it) is thus connected into a linear sequence, it can be dramatized by invoking the Drama Manager with the *render* command.

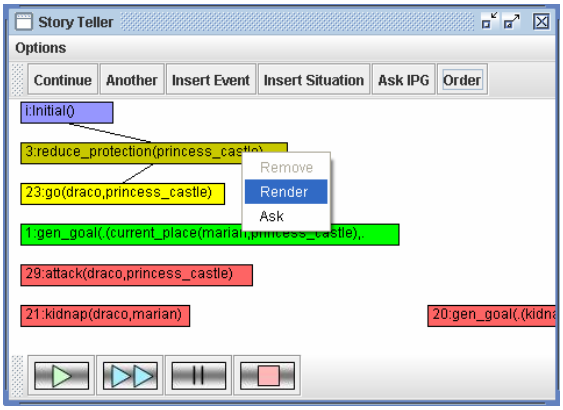


Figure 4: Plot manager interface.

As mentioned, story generation and dramatization are treated as separated processes. During dramatization, virtual actors in a 3D-world perform actions without user (player) interference. Plot dramatization can be activated for the exhibition of the final plot or the partial plots. The simulation occurs in continuous real-time and the duration of an event rendering task is not previously known.

During the dramatization, the high level control of actions each actor is supposed to perform is done by the Drama Manager. The Drama Manager translates symbolic events into fully realized 3D visual graphical atomic actions and assign then to specific actors at specific times, guaranteeing the synchronism and logical coherence between the intended world and its graphical representation. It continuously monitors the representation process, activating new tasks whenever the previous ones have been finished. There is a near 1-to-1 mapping among events and graphical actions performed by the virtual actors.

During the performance of an event, actor's low-level planning is used to detail the tasks involved in each event. By means of a built-in finite state machine, actors may select ways to perform the actions being assigned by the drama manager. Possible actions are walk, avoid obstacles, fight, kill, marry and others.

The process of capturing the scenes and record animated representation in real-time is done by a camera agent. As the story unfolds and events are dramatized, the camera, implemented as a reactive agent, has a role of capturing actions performed by actors (same approach adopted by Charles [2002]). The camera continuously monitors the actions and respective actors involved in order to position himself and select a reasonable angle of view to record the graphical representation.

As illustrated in Figure 5, Drama Manager sends simultaneously actions to the virtual actors as well to the camera. This approach has serious drawbacks. For example, frequently actions start before camera had time enough to contextualize scene, an important cinematography rule to allow a comprehensive understanding of the take. Moreover, user has no availability to intervene the way scenes are being recorded.

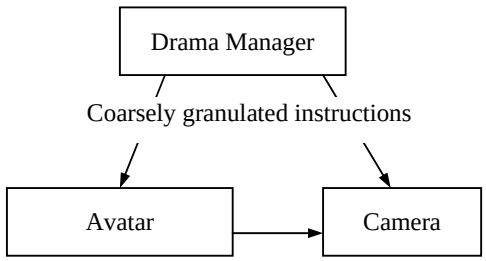


Figure 5: Dramatization architecture

The camera commonly uses sets of predefined viewpoints, which are automatically specified according actor's positions and scenario configurations. To render the events, both camera and avatars receive simultaneously required information. As the avatar starts performing the action, the camera follows its movements without concerning scene and filmmaking constraints. For example, in a Go(Draco, Princess_Castle) event, Draco starts walking from its current location to anywhere he's supposed to go and the camera keeps up with its movement, with a predefined relational position. The camera does not use contextualizing takes; when the recording begins, the actor was already walking towards the target.

The actors are implemented as partially reactive agents. They have the ability to choose actions that

best fulfils events dramatization and to communicate with other agents whenever necessary. This is not an appropriate approach when the user has the ability to change the story script because action selection is coded into the actor's finite state machine and cannot be changed in real-time.

4.1. The new Architecture

In the new approach, both actors and camera lose part of their autonomy in order to follow detailed instructions given by the director. To do such, each event has a set of camera positions that can be used in the shoot, and a script to be followed by the actors and camera.

To improve the dramatization process and achieve a high level of interactivity, a graphical interface is available to edit the story's details - users can set audio parameters, create idioms, edit and delegate actions and develop a conversation between the actors.

In the core of the new architecture there is a director. Figure 6 illustrates in more detail the new relationship among the Director (old Drama Manager), actors and the camera. Messages exchanged between components are synchronized to allow, for example, that camera receives in advance the event the avatar is supposed to perform.

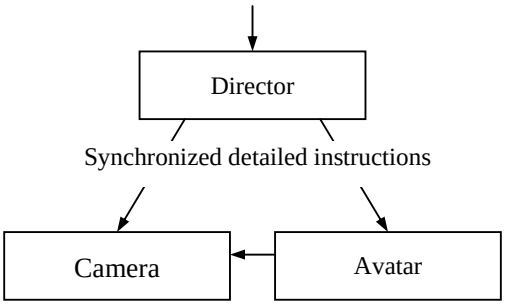


Figure 6: Relationship among Director, actors and camera.

5. The System Architecture

Although a film can be seen as a set of individual motion pictures, it is often helpful to think of a film as having a structure. At the highest level, a film is a sequence of scenes, which captures determined events. The scenes are composed by shots. Shots are fractions of time that the camera records continuously.

As the rendering process of a storytelling environment approaches filmmaking, our system has similar architecture. We have proposed some film basic elements that are divided into **shots**, **scenes**, **facts** and a **film configuration**. The correlation between these elements is shown in Figure 7.

The primitives that compose each shot are packed into a shot data structure that helps the render to captures scenes. This shot structure contains variables such as camera position, angle of camera, duration of the shot and movement.

The scene data structure contains all information defined by the user to make the shots. Fact that composes the scene and the information about how each shot must be filmed are the parameters defined in this level.

Facts are the pillar of the scene: they contain the information that tells what and why the actors should do. The main advantage of separating the scenes from facts is that we can provide an interface to edit the facts without concerning (at least at this level) about the scene filming. So, each event generated by IPG are translated into a fact

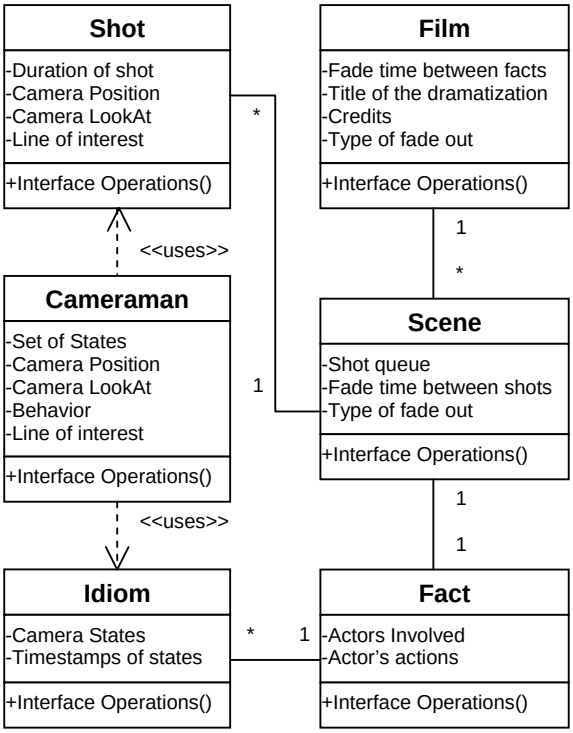


Figure 7: System architecture

The last one component, a film configuration, is a set of high level parameters used to enhance visual presentation. Features like title presentation, credit presentation, fade-out time between facts and type of fade-out are available for editing by users.

Next section presents our implementation of the camera module as a cameraman. Section 5.2 presents how idioms are coded and its relationships with the camera module.

5.1 Camera module as a Cameraman

Once cinematographic expertise is defined as a low level set of constraints that changes with the flow of the time, we used a cameraman, an intelligent agent that knows how to behave through time, given specific parameters. Our agent was implemented based on finite state machines, a good solution given the characteristics of our problem.

Each state of the cameraman has three main parameters: the initial camera placement, the camera distance and how the camera will behave through the shooting. For example, the possible init placements, as described in Figure 2, are: External, Parallel, Internal, Apex, Reverse Internal, Reverse Parallel and Reverse External. Reverse placements are faced to the ending of the interest line.

The possible sets of camera distance were defined by cinematographers based upon cutting heights. The main rule in cutting heights is that the frame lines do not cut the actor's primary joints, since this has a strange look on screen. Primary joints include the neck, waist, knees, and ankles. Our possible sets of camera distances:

- Extreme close-up: cuts-up at the neck;
- Close-up: cuts-up at the chest;
- Medium-view: cuts-up at the crotch;
- Full-view: Full view of the actor;
- Long view: Distant perspective.

Camera behaviors have been implemented based on the work of Christianson [1996] (Figure 8). They are as follows:

- Static: Object and the camera stand still.
- Go-by: Camera follows up the object, preserving a static angle between the line of interest and the camera.
- Panning: Camera follows up the object, keeping it near the center of the screen as the camera turns.
- Tracking: Object always will be in the center of the screen, and the camera will follow it preserving a static angle of 90 degrees between the line of interest and the camera.

The combination of these configurations gives us the set of all the states. Some of these combinations are normally not used – as example a reverse external close-up go-by state: it is a little awkward having the main object from a backed face close-up view leaving out the viewport without being followed.

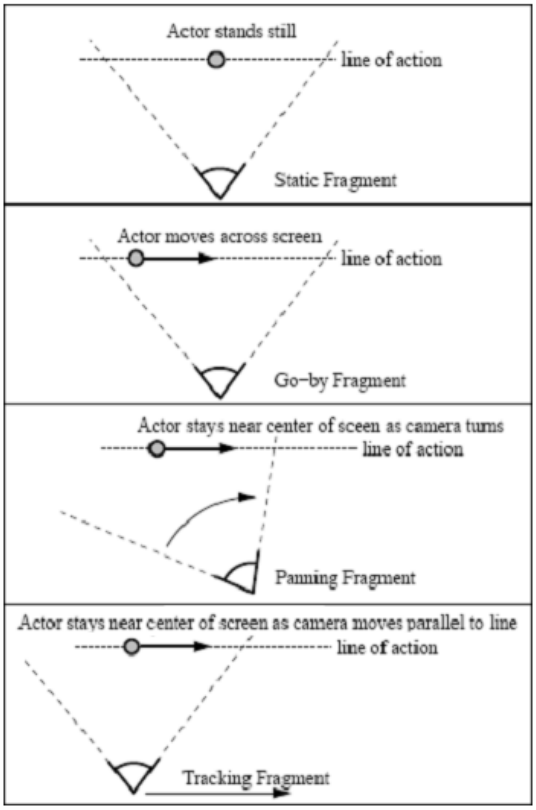


Figure 8: Possible camera behaviors, extracted from [Christianson 1996]

5.2 Idioms

Idioms are vastly used by many authors in camera systems [Halper 2000]. With this method, cinematographic knowledge is easily coded into constraints to assure good shots. We implemented idioms by sets of camera states: since the cameraman provides a simple high level interface that abstract camera placements and behavior, the process of making an idiom is quite simple. We must order the camera states according a timeline and modify the position of the line of interest. Our approach is similar to the Declarative Camera Control Language (DCCL) (Christianson 1996). We adopted the following syntax:

idiom_name (arguments)
(timestamp): state (initial Placement, distance of the line, behavior)
(initial point of the line, final point of the line).

As an example, consider the creation of an idiom for the event where a character goes to a place. As one can see, we can set a number of timestamps for each event that are used to change the cameraman's states. In this case, the timestamps are based upon the total time of the scene, which is calculated measuring the time of all the actions of the actors involved in the current filming sequence.

Go1(character CH, placement PL):
(0): state(External, Medium view, Tracking) (CH, PL)
(totalTime*1/4): state(Apex, Distant view, Go-by) (CH, PL)
(totalTime*3/4): state(Internal, Medium view, Tracking) (CH, PL)

In this configuration example, the idiom will start with the initial placement (timestamp equals to zero) of an external camera, a camera distance of a medium view and a tracking behavior. The first (the character) and second (the placement) arguments passed to the state are, respectively, the initial and the final points of the line of interest.

To simplify the process of creating the takes, each fact has one or more idioms mapped into it. If a fact has more than one idiom assigned, the idiom will be picked randomly among the possible idioms. The advantage of assigning more than one idiom to a fact is that the fact can be dramatized of different ways at different executions.

6 Script Edition System - SES

Storytelling applications work with events. In our test scenario, they are generated by IPG, and, generally, are a single objective phrase that assumes one goal. "Go(Draco, Princess_Castle)", "Kidnap(Draco, Marian)", "Fight(Brian, Draco)" are examples of events that can be generated.

However, these objective phrases, just by themselves, leave a great margin for cinematography creation. As an example, consider the event where dragon kidnaps the princess – in the simpler approach, the dragon could just go to the victim and take her to its prison. Considering a more sophisticated and realistic representation, the victim could refuse of being kidnapped and decided, in vain, to fight with the dragon.

To allow a greater user interaction and a more sophisticated dramatization, each event in the IPG has a correspondent fact that describes it as a set of simple actions that each actor *knows* how to perform. We use the tag "Script:" to indicate all actions that an event has and cannot be altered.

The role of the SES is to provide a simple high level interface to users, allowing them to insert some details of the story before the dramatization process begins. The tag "User input:" is used to mark the actions specified by the user. The tables of actions that users can insert are displayed in the Tables 1, 2 and 3. To insert new actions on these tables we need to implement respective functionalities in our dramatization engine.

The following example illustrates a script that can be edited with the SES. This script is an extended version of the Fight(Brian, Draco) IPG's event. In this example, Brian must first go to Draco's place and fight with him. As one can see, the user has inserted three actions, enriching the dramatization process.

Fight(Brian, Draco)
Script: **Walk(Brian, Place)**
User input: Play_sound(FightSong)
User input: Talk(Draco, "You foolish! How dare you to challenge my POWER?!", 12)
User input: Talk(Brian, "You naughty creature! Thou shall taste my sword!", 10)
Script: **Fight (Brian, Draco)**

As SES gives the user a high interactivity, it does not prevent a logical incoherence in the story. The user may deliberately delegate actions without consider the main goal of the event, instructing, for example, Draco to kiss Brian before they start fighting.

Table 1: Collective actions that actors can perform.

Action	Arguments	Description
Kiss	actor1, actor2, time	The actor1 will kiss actor2 during an amount of time.
Fight	actor1, actor2, time	The actor1 will fight with actor2 during an amount of time.

Table 2: Environment actions

Action	Arguments	Description
Play	File	Play sound file once
Play_loop	File	Play sound continuously
Stop		Stops the sounds that's being played

Table 3: Single actions that actors can perform.

Action	Arguments	Description
Walk	Actor, place	The actor will walk to a given place
Laugh	Actor, time	The actor will start to laugh during an amount of time
Pain	Actor, time	The actor will simulate pain during an amount of time
Cry	Actor, time	The actor will start to cry during an amount of time
Dance	Actor, time	The actor will start to dance during an amount of time
Talk	Actor, msg, time	The actor displays a message during an amount of time.
Think	Actor, msg, time	The actor's thoughts will be expressed through a message that will be displayed during an amount of time.

Environment actions are those who are performed by the environment. They don't need to be particularly filmed (so doesn't need an idiom mapping) and they can happen concurrently with single and collective actions. Single actions are performed by one actor and need to have an idiom mapping. Collective actions are similar to single actions, but their line of interest must be defined in function of the actors involved in the shooting.

7. Dramatization Process

The dramatization process is divided in two defined stages: Planning and Execution. In the planning stage, all the initialization are made. With the set of facts, we first create a queue of facts in some chronological order. Since each fact has a set of idioms (or at least one idiom), a mapping from one to one is made. With this, we have the parameters necessary to pre-create all the scenes and shots of the dramatization. Each scene has a queue of shots; the shots are initialized by defining their duration time and its initial line of interest. When all the scenes are pre-initialized, a scene sequence is made.

In the execution stage, the dramatization is properly done. The renderer takes a scene from the scene sequence and then a shot from the shot queue. The shot has an update function that calls the cameraman to update position and orientation of the camera (that is defined by the cameraman's behavior, which is given by the idiom of the current fact that is being dramatized), and an update function that invites all the actors involved to perform their actions and update their positions.

8 Conclusions and Future Work

This work proposes a new architecture for specifying parameters to enrich dramatization process over an existing storytelling environment called LOGTELL. As presented, the original architecture was rigid and actors and camera, during dramatization, were limited to execute to pre-programmed actions on their finite state machine.

With the inclusion of a virtual director, both actors and camera lose part of their autonomy in order to follow not more high level actions, but detailed instructions that are originated from user intervention plus cinematographic expertise like idioms and camera settings.

Techniques and resources upon 3D models could not be fully implemented because the limitations of our test environment. Possible animations are hard-coded on a key-framing animation file. Animations are limited to basic actions like walk, fight, stand and die. With our proposed architecture, more realistic and emotive actions like kiss, hug, among others, are not allowed to be graphically visualized. Even not implemented, one can see the feasibility of implementation of these ideas. These points here presented are focus for future works.

We are also planning to incorporate emotions and facial expressions to the dramatization. To allow this, we must focus specially at the 3D models, because at the level of the Scene Creation System, simple parameters can provide the required information.

Due to limitations in our test environment, currently we have implemented only the SES interface, through were user can enter/edit commands to enhance visual presentation. It is implemented in Java, the same language were used to program the Plot Manager. This way, we added an extra button to the popup menu of Plot Manager to invoke this new interface. Those new commands use a modified version of the protocol that communicates Plot Manager to the Director (old Drama Manager implemented in C++).

References

ARIJON, D., 1976. Grammar of the Film Language. Communication Arts Books, Hasting House, Publishers, New York.

BARROS, M., L. 2004. Incluindo elementos da narrativa em Interactive Storytelling. Trabalho de conclusão de curso. Unisinos.

CAVAZZA, M., CHARLES, F. AND MEAD, S., 2002. Character-based interactive storytelling. IEEE Intelligent Systems, special issue on AI in Interactive Entertainment, 17(4):17-24.

CHARLES, F., LUGRIN, J., CAVAZZA, M. AND MEAD, S., 2002. Real-time camera control for interactive storytelling. In: GAME-ON, London, UK.

CHRISTIANSON, D. B., ANDERSON, S. E., HE, L., COHEN, M. F., SALESIN, D. H., WELD, D. S., 1996. Declarative Camera Control For Automatic Cinematography. In Proceedings of AAAI '96, 148-155.

CIARLINI, A., POZZER, C. T., FURTADO, A. L. AND FEIJÓ, B., 2005. A logic-based tool for interactive generation and dramatization of stories, ACM SIGCHI International Conference on Advances in Computer Entertainment Technology – ACE 2005, Valencia, Spain, 133-140.

CIARLINI, A., VELOSO, P., AND FURTADO, A., 2000. A Formal Framework for Modelling at the Behavioural Level. In Proc. The Tenth European-Japanese Conference on Information Modelling and Knowledge Bases, Saariselkä, Finland.

COURTY, N., LAMARCHE, F., DONIKIAN, S., AND MARCHAND, E., 2003. A Cinematography system. In ICVS 2003, LNCS 2897, pp. 30–34, 2003.

HALPER, N., HELBING, R., STROTHOTTE, T., 2001. A camera trade-off between constraint satisfaction and frame coherence. in Eurographics, volume 20.

HALPER, N. AND OLIVER, P., 2000. CAMPLAN: A Camera Planning Agent. In *Smart Graphics". Papers from the 2000 AAAI Spring Symposium (Stanford, March 20–22)*, Menlo Park, AAAI Press, pages 92–100.

HE, L., COHEN, M., AND SALESIN, D. 1996. The virtual cinematographer: A paradigm for automatic real-time camera control and directing. In Proceedings of the ACM SIGGRAPH '96, 217-224.

GRASBON, D. AND BRAUN, N., 2001. A morphological approach to interactive storytelling. In: Fleischmann, M.; Strauss, W., editors, Proceedings: CAST01, Living in Mixed Realities, Sankt Augustin, Germany, p. 337-340.

MATEAS, M. AND STERN, A., 2000. Towards integrating plot and character for interactive drama. In: Dautenhahn, K., editor, Socially Intelligent Agents: The Human in the Loop, AAAI Fall Symposium, Technical Report, p. 113-118.

SCIENTIFIC AMERICAN, 2000. Special issue on digital entertainment, 283 (5), November.

SPIERLING, U., BRAUN, N., IURGEL, I. AND GRASBON, D., 2002. Setting the scene: playing digital director in interactive storytelling and creation. Computer and Graphics 26, 31-44.

YOUNG, R., 2000. Creating interactive narrative structures: The potential for AI approaches. In: AAAI Spring Symposium in Artificial Intelligence and Interactive Entertainment, Palo Alto, California. AAAI Press.

Towards Consistency in Interactive Storytelling: Tension Arcs and Dead Ends

Leandro Motta Barros
Universidade do Vale do Rio dos Sinos
São Leopoldo, RS, Brazil

Soraia Raupp Musse
Pontifícia Universidade Católica do Rio Grande do Sul
Programa de Pós-Graduação em Ciência da Computação
Porto Alegre, RS, Brazil

Abstract

Interactive Storytelling (IS) systems are an emerging class of interactive entertainment applications with emphasis in narrative aspects. One of the approaches used to develop IS applications is based on planning algorithms. This paper describes two mechanisms that can be introduced to planning-based IS in order to solve two problems that are inherent to this approach. The first is a technique to control the pace of story evolution, so that the generated stories follow a specified tension arc. The second is a method that can avoid story dead ends by intervening in the story in a way that is justified with previous story events. These mechanisms help to produce a more enjoyable user experience, since their use tends to result in stories with higher narrative consistency. Results from experiments with users suggest that the proposed mechanisms improve both of the aspects being addressed by this work.

Keywords:: Interactive Storytelling, narratives, planning, riddles, tension arcs, story dead ends

Author's Contact:

leandromb@unisinos.br
soraia.musse@pucrs.br

1 Introduction

The fabulous commercial success attained by several technically advanced computer games has stimulated the further development and adoption of new technologies applied for the interactive digital entertainment. Perhaps this is more clearly visible in the area of Computer Graphics, but there are also evident advancements in areas like Artificial Intelligence (AI) and Computer Networks. On the other hand, a less technical and more conceptual analysis of the games developed in the past decades, reveals that most of them are designed around the same concepts: hand-eye coordination, puzzle-solving and resource management [Crawford 2004].

Several researchers believe that it is possible to go beyond this, and that the exploration of aspects neglected by today's games can originate new forms of interactive digital entertainment. Among these researchers, Janet Murray [Murray 1997], Andrew Glassner [Glassner 2004] and Chris Crawford [Crawford 2004] believe that narrative and interactive aspects can be combined in order to generate a new media for storytellers and a new form of artistic expression. Transforming this vision into something concrete requires us to deal with a new set of challenges. The field that deals with the challenges required to create interactive applications capable to generate consistent narratives is called Interactive Storytelling (IS).

An IS system can be seen as an agent that tell stories, and that adapt the stories as it interacts with the player. One of the most important challenges in the field of IS is how to model this storyteller agent such that it achieves a good balance between interactivity and narrative consistency. In other words, it is desired that the system actually takes the player actions into account, but still generates interesting stories, capable to please the player.

Researchers on the field of IS are exploring different approaches in order to achieve this goal. One of these approaches is based on the use of planning algorithms to generate the sequence of events that compose the story. Planning-based IS has some interesting characteristics, but it has one fundamental drawback: general purpose planning algorithms are created in the context of AI, without any consideration of narrative aspects. Consequently, planning algorithms *per se* will not be able to generate consistent stories: we have to explicitly introduce mechanisms to enforce this.

In this paper, we present two of such mechanisms. First, we describe in detail a method that can be used to make the generated stories follow a tension arc specified by the author.[1] This method is designed for stories following the Riddle master plot [Tobias 1993]. Second, we introduce a way to subtly intervene in the stories in order to avoid some story dead ends.

The next session presents some work related to ours. Section 3 provides an overview of Fabulator, our Interactive Storytelling prototype, in which the ideas presented in this paper were implemented. The main contributions of this paper are in sections 4 and 5, which describe the two mechanisms mentioned in the previous paragraph. This is followed by the presentation of results (Section 6) and, concluding the paper, some final remarks.

2 Related Work

The similarities between plans and stories have been explored since the early systems for non-interactive story generation, like UNIVERSE [Lebowitz 1985]. In this work, plot outlines resembling soap operas are generated by using planing algorithms and libraries of characters and plot fragments.

Our work is based on the use of one particular type of planning systems, namely, STRIPS-like planning systems. This approach was introduced in a work by Charles *et al.* [Charles et al. 2003], in which planning was used to create sequences of actions allowing the story characters to pursuit their goals. In a previous work [Barros and Musse 2005], we built on these ideas, concentrating on methods that could improve the narrative consistency of the IS model described by Charles.

The work by Young and collaborators [Riedl et al. 2003], which is also based on planning, observed that the player is able to perform actions that are totally incompatible with the plot created by the planning algorithm. For example, the player may try to kill a character that is expected to play a fundamental role in the upcoming story events. Young calls these cases "exceptions", and proposes intervention mechanisms that can avoid them. In extreme cases, the idea of intervention consists in making the player action fail (*e.g.*, making the gun misfire). Exceptions are closely related to the notion of "dead ends" that we deal with in this paper. Indeed, the intervention mechanisms proposed by Young can be used to deal with dead ends, but we believe that the approach we introduce in this paper is more subtle, thus less damaging to the narrative consistency.

The architecture described by Magerko [Magerko 2005] includes a probabilistic player model, used to predict the player behavior, so that the system can anticipate his or her actions. Our own work currently does not uses a player model, but as we discuss later, a model like the one described by Magerko could help to improve the performance of the method we propose for foreseeing story dead ends.

Finally, Mateas and Stern [Mateas and Stern 2003] have taken a lot of care to generate highly consistent stories in the interactive drama Façade. In particular, they try to ensure that the generated stories

[1] A poster with a short description of this specific point has been published previously [Barros and Musse 2007]. Here, we provide additional details about it and present more results.

follow a certain tension arc. Our work also presents a mechanism that aims to make the generated stories follow an author-defined tension arc, but our work (which is in the context of planning-based IS) is considerably different.

3 Architecture Overview

Before detailing the two mechanisms that are our main contributions, we present an overview of Fabulator, our prototype IS system. A more complete description of this architecture can be found in [Barros and Musse 2005]. The player controls a single character, the story protagonist. All other characters are NPCs (Non-Player Characters), controlled by the system. A story can be seen as a sequence of actions performed by characters that can transform the world in a way imagined by the storyworld author. This sequence of actions is modeled as a plan created by a planning algorithm. Actions performed by the player may render the current plan invalid. Whenever this happens, a new plan is created, so that the story is adapted to the player's actions. We can divide the architecture in three main modules, as shown in Figure 1.

The first module, *Actions Selection*, is responsible for generating plans composed of actions capable to transform the world from its current state to a desired state. In our current implementation, this sequence of actions is generated by the Metric FF planning system [Hoffmann 2003].

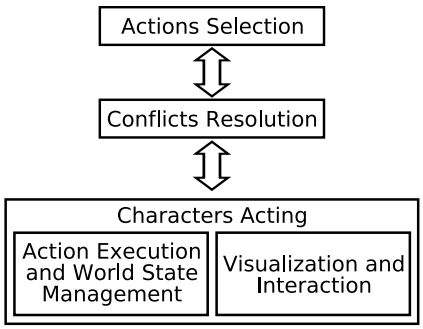


Figure 1: *Our prototype's architecture.*

The set of actions available to the planning algorithm is described in a STRIPS-like language (actually, the language used in Fabulator incorporates all ADL extensions described in [Russell and Norvig 2002]). Essentially, actions have two components: a prerequisite and an effect. The prerequisite is used to verify whether the action is usable given the current world state. The effect defines how the world state shall change when the action is actually performed. Figure 2 shows an example of action definition.

```
GivePresent (Character Giver, Character Receiver,
             Thing Present)
   PREREQUISITE
      AND (
         NOT EQUAL (Giver, Receiver),
         HAVE (Giver, Present),
         EXISTS Place P (
            AND (
               At (Giver, P),
               At (Receiver, P))))
   EFFECT
      AND (
         HAVE (Receiver, Present),
         NOT HAVE (Giver, Present),
         IF Likes (Receiver, Present)
         THEN Likes (Receiver, Giver))
```

Figure 2: *An example of an action described in the language used by Fabulator.*

Since each storyworld requires a different set of actions, the definition of the actions is a task that must be performed by the storyworld author. In fact, the definition of actions is the most important and time-demanding task for the author.

The *Characters Acting* module is responsible for making the characters act, that is, to make them execute the actions determined by the *Action Selections* module (for NPCs) or issued by the player (for the protagonist). As part of this responsibility, this module keeps and manages a set of predicates representing the current world state. Concerning visualization, our prototype displays the story to the player with animated characters in a three-dimensional environment, implemented using freely available libraries: Open Scene Graph (`http://www.openscenegraph.org`), CAL3D (`http://home.gna.org/cal3d/`) and CEGUI (`http://www.cegui.org.uk`). The player executes actions by using the mouse. Clicking in a character or object brings a list of actions related to what was clicked.

The third and final module is the *Conflicts Resolution* module. Its purpose is to deal with conflicts that may arise when a new plan must be generated. In these cases, the new plan may determine that some character must perform some action that is different from the action it is currently performing. The role of this module is to chose if the character will finish its current action before starting the new one, or will interrupt the current action and start the new one immediately. This decision is based on a priority value associated with every action.

Our prototype cannot be considered a complete IS application. It still lacks several important features, but it is a working IS prototype that has been evaluated by a group of users, and can be used to test and evaluate novel ideas in the field of Interactive Storytelling. Figure 5 shows screenshots of Fabulator.

4 Tension Arcs

It is not possible to create an interesting story without obstacles between the protagonist and its goals [Tobias 1993]. These obstacles are the source of tension of a story, and tension is carefully controlled by story authors: they make it raise or lower as the plot unfolds, depending on how they want to make the audience feel.

The level of tension in a story in function of time results in a curve called *tension arc*. The most frequently used tension arc is the Aristotelean Arc (composed of rising tension, climax and denouement), but variations are possible. Zagalo [Zagalo et al. 2004], for instance, mentions that many recent movies superimpose smaller secondary tension arcs over the main arc. In other words, the decision on how the tension arc of a story should be is a competence of the author of the story.

In this work, we are dealing with stories that follow the Riddle master plot described by Tobias [Tobias 1993]. This encompasses all stories in which there is a mystery to be solved, like detective stories ("whodunits"). Unfortunately, Tobias is very brief when discussing tension or tension arcs for this kind of story. Actually, he only mentions that "the tension of your riddle should come from the conflict between what happens as opposed to what seems to have happened."

Next, we present a model for the tension arc in riddle stories that is based on this observation by Tobias. Then, we describe a method that can be used to make the stories generated by the system follow a tension arc specified by the author of a storyworld.

4.1 Modeling Tension Arcs

Our model of tension arcs is based on the idea that the tension in riddle stories raises as new clues are discovered or revealed, making the knowledge that the player has about the events (what seems to have happened) approach to the truth (what actually happened). The moment in which the player needs just the final clue to solve the riddle is the tension apex. The story ending coincides with the instant in which the player's knowledge reaches what actually happened, that is, it is the moment in which the riddle is solved. Thus, in the proposed model, the player knowledge about the riddle and the tension are equivalent.

These concepts can be formalized as follows. The tension arc of a story is a function of time, designated by $K(t)$. So, if the player starts the story without any clue that may help to solve the riddle, the player's knowledge (and therefore the tension) would be zero, *i.e.*, $K(0) = 0$. Whenever a clue is revealed to the player, the value

of K is incremented (and this is the only way by which K can change). The amount by which K is incremented depends on the importance of the clue for the solution of the riddle: the more important the clue, the larger is the increment. Generically, when a clue c is revealed, K is incremented by k_c. The values of k for each clue are to be chosen by the storyworld author. So, if a clue c is discovered at an instant t, the value of K is incremented by k_c at this instant. It worths mentioning that, in this model, $K(t)$ is a non-decreasing monotonic function, since the player knowledge never shrinks. See Figure 3 for an illustration of these ideas.

4.2 Respecting Tension Arcs

Before introducing a mechanism that allows the generated stories to follow a desired tension arc, it is necessary to define how is this desired arc. Following the ideas presented above, the desired tension arc is a function of time, defined by the author of the storyworld and designated by $K^*(t)$. In principle, $K^*(t)$ could be any non-decreasing monotonic curve, but our current implementation requires it to be composed of increments at discrete time instants (that's why the desired tension arcs in figures 3 and 6 have a "stairs-like" appearance).

Given the definitions of the actual ($K(t)$) and desired ($K^*(t)$) tension arcs, we can define a new function, $D(t) = K^*(t) - K(t)$, that measures the discrepancy between these two curves (again, see Figure 3).

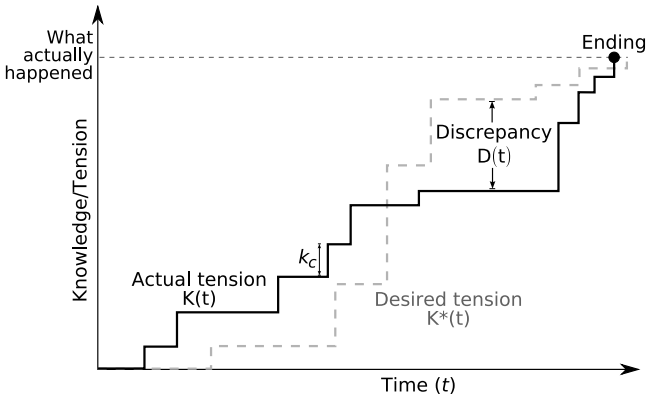


Figure 3: *Model of tension arc.*

Ideally, there should not be any discrepancy between the actual and the desired tension arcs, that is, we would like to have $D(t) = 0$ for all possible values of t. In practice, however, there always will be some difference between the two arcs. To quantify the "total discrepancy" observed in a story, we introduce a new value, E, defined as

$$E = \int_{t=0}^{T} |D(t)| \,,$$

where T is the total story duration. This way, the intuitive notion of respecting a tension arc can be formally defined as minimizing E.

In our model, the value of $K(t)$ can only change when a clue is revealed. Therefore, if the current tension level is above the desired ($K(t) > K^*(t)$), we need to somehow slow down the pace at which clues are discovered. Likewise, if the current tension level is below the desired ($K(t) < K^*(t)$), something must be done to help the player finding clues more quickly. Doing this, the discrepancy $D(t)$ will tend reduce and, consequently, E will be minimized. In order to achieve this, we dynamically change the level of participation that NPCs have in the discovery of clues (for instance, if the player is having difficulties to find clues, NPCs should work more to help). We do this by varying the cost of actions performed by NPCs (denoted c_{NPC}), as described below. (The cost of the protagonist actions is constant: $c_{\mathrm{prot}} = 100$.)

For any given instant t (when a discrepancy $D(t)$ is observed), we assume that there exists an ideal value for the cost of NPC actions (we denote this value by $c^*_{\mathrm{NPC}}(t)$). This value is considered the ideal in the sense that using it will incur the generation of a plan in which the NPCs will act in the right amount to bring the story back to the desired tension level. *A priori*, we cannot know how the discrepancy, the action costs and the level of participation of NPCs are related. So, we conducted several experiments with the Metric FF planning algorithm and the storyworld used in our tests (*Ugh's Story 2*, described in Section 6). We generated several plans using various values for c_{NPC} and analyzed them. This allowed us to find some good values for c_{NPC} given some values of $D(t)$. Later, we found that these pairs of good values could be well fitted by the function

$$c^*_{\mathrm{NPC}}(t) = \lfloor -1.875 D(t) + 75 \rfloor \,,$$

which was used in the experiments related in this paper. (The floor operator is used because our planning system accepts only integer costs.) As an aside, the value 75 in the equation above can be seen as a "difficulty level". Increasing it will make the NPC action costs higher, and thus will make them less helpful.

In runtime, our system compares the value of the NPC action cost used to create the current plan to the value of $c^*_{\mathrm{NPC}}(t)$ (this is done every 15 seconds). Whenever this difference gets over 25%, or if the absolute difference gets above 20, a new plan is generated using the current value of $c^*_{\mathrm{NPC}}(t)$ for the NPC actions cost.

5 Dealing with Dead Ends

The player has the ability to modify the world state by commanding the protagonist to execute actions in the story. He is therefore capable of leading the story to a dead end, that is, he can change the world state to one from which there exists no plan capable to bring the story to the desired ending. The intervention mechanism proposed by Young [Riedl et al. 2003] (see Section 2) is efficient to solve this problem in the sense that it can completely avoid dead ends, but it has one drawback: the player actions leading to a dead end are effectively ignored by the system in the moment that they are executed. This can be frustrating, because the player expects that his actions actually influence the story.

The alternative we propose here cannot totally eliminate the problem of dead ends, but, when applicable, we believe it provides a better solution, because the intervention does not occur at the very last moment, and it is justified by previous story events. The idea is composed of two steps: foreseeing dead ends and acting to avoid them.

5.1 Foreseeing Dead Ends

The player has the freedom to execute the actions he wishes, but he must chose his actions from a finite repertoire of actions specified by the storyworld author during the authoring process. So, while it is not possible know what the player *will* do, it is possible to know everything he *can* do.

This allows us to expand the tree of world states to where the player can lead the story, as seen on Figure 4. In this figure, the current world state is s_0, at the tree root. In this state, the player has as action options $a_1, a_2, \ldots, a_n$. If he chooses a_2, the world state will change to s_2, and he will have a new set of actions to choose from $(a_1, a_2, \ldots, a_m)$.

Therefore, dead ends can be detected by expanding some levels of this tree and testing, for each expanded state, if the planning algorithm is capable to create a plan that transforms this state into a goal state. Figure 4 shows a case in which the story will reach a dead end if the player executes the sequence of actions $\langle a_2, a_1 \rangle$.

Concluding, we would like to discuss a limitation of the dead end foreseeing method we described. The branching factor of the tree that must be expanded is typically large (because there are many actions available for the player at any state), so, contrary to what would be desired, it is impossible to expand many levels. Despite this limitation, we believe that the proposed method is an interesting contribution, since this limitation can be minimized in the future. An interesting possibility in this regard would be using a user model to estimate the most probable future user actions, and expand only the branches that have more likelihood of happening.

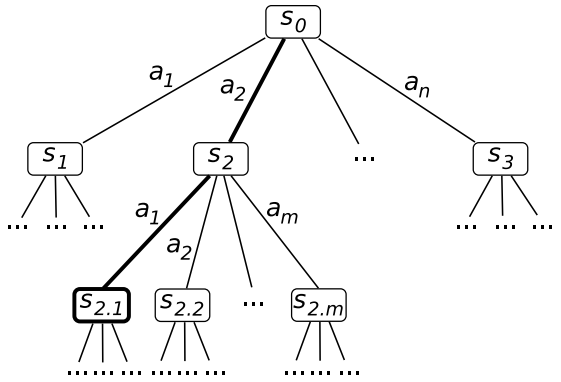


Figure 4: *Foreseeing dead ends. State $s_{2.1}$ represents a dead end.*

Magerko [Magerko 2005] has already shown that user models can be used to foresee the user behavior in IS applications.

5.2 Avoiding Dead Ends

Once a certain state is identified as a dead end, we must somehow intervene in the story to make this state unreachable. This can be done by two distinct ways:

1. Changing something in the world state that precludes the execution of one of the actions that lead to the dead end.
2. Modifying the current world state so that the execution of the "dangerous action sequence" will no longer result in a dead end.

But care must be taken when intervening in the story, because if the intervention is "too explicit", the narrative consistency can be harmed. Bearing this in mind, we propose the introduction of two types of actions: *pretext actions* and *intervention actions*.

Pretext actions are actions that don't have any prerequisite and, therefore, can be executed anytime. The only purpose of pretext actions is to introduce in the world state some predicates that serve as prerequisites for intervention actions. The intervention actions, as the name suggests, are the actions actually used to intervene in the story to prevent the player of reaching a dead end. For example, a pretext action could be defined like below.

Action: *Storm*
Prerequisite: None
Effect: *StormHappened*

Next, a number of intervention actions with *StormHappened* as prerequisite could be defined. For example:

Action: *FallBridge*
Prerequisite: *StormHappened*
Effect: ¬*HouseReachable*

Action: *FallIll (character)*
Prerequisite: *StormHappened*
Effect: *IsIll (character)*

Action: *ExtingsuishFire (building)*
Prerequisite: *StormHappened*
Effect: ¬*InFlames (building)*

If in a moment in the story it is detected that a dead end will be reached if the player visits his house, and if the *Storm* pretext action has been previously executed, it will be possible to execute the *FallBridge* intervention action that will avoid the problematic situation with the pretext that the bridge has fallen due to the storm.

Technically, it would be possible to execute a pretext action with a specific intervention action already in mind, just after a dead end is detected. This would make the intervention too explicit to the player, though. We believe that better results are achieved if the pretext actions are executed at random times in the stories. Naturally, this approach increases the chance of the proposed mechanism being unable to avoid a dead end. For these cases, the intervention mechanism proposed by Young [Riedl et al. 2003] could be used as a last resort.

6 Results

The methods described in the previous sections were implemented in our prototype and actually tested with a storyworld named *Ugh's Story 2*, an extended version of the original *Ugh's Story* introduced in [Barros and Musse 2005]. The story passes in Neandertown, a small village inhabited by cavemen. Neandertowners used to worship a statue located on an altar near the caves in which they live. One day, Alelugh, the local priest, was found unconscious near the altar, hit by a club, and the statue was stolen. The player plays the role of Ugh, a detective caveman hired by Alelugh to investigate this case and discover the author of this crime. Two other characters participate in the story: Egonk and Brokung (both are Neandertowners). As the story unfolds, the player discovers that both Egonk and Brokung had reasons for committing the crime. Egonk is very egotist and selfish, and could have stolen to have the statue just for himself. On the other hand, Brokung is bankrupt and could have stolen the statue to sell it. As more clues are revealed, the player eventually finds out that Brokung stolen the statue and sold it to Salesorg, the traveling salescaveman. All clues that can be found are listed in Table 1. Some screenshots of the storyworld are shown in Figure 5.

c	k_c	**Clue**
1	4	The statue was stolen.
2	5	Brokung had loan request rejected.
3	6	Brokung is deeply indebted.
4	5	Brokung admitted having debts.
5	5	The statue worths a lot of money.
6	5	Egonk has a personal statuette.
7	5	There is a "suspect" fire starter.
8	6	Egonk stolen the fire starter he envied.
9	4	Egonk said he doesn't borrow his things.
10	4	Brokung said that Egonk is egotist and selfish.
11	4	The crime happened yesterday, at noon.
12	5	Brokung and Egonk were hunting together at the crime time.
13	4	There is a large carcass of a recently hunted animal in the forest.
14	7	Meat and leather were sold to Salesorg.
15	6	Egonk prepared the meat and the leather while Brokung went to negotiate with Salesorg.
16	7	Salesorg bought the statue.

Table 1: *Clues in* Ugh's Story 2*. The column k_c shows the values used for the k parameters (introduced in Section 4.1).*

Evaluation was made by letting a total of five users play *Ugh's Story 2*. Before playing, each user received a brief overview of the crime they were about to investigate and instructions on how to play. While playing, their reactions have been observed, and log files were generated, containing all relevant events necessary to review their experience. Finally, after playing, each user was interviewed, so that we could get some feedback on their general feelings and opinions.

6.1 Total Discrepancy

One of the information recorded in our log files is the total discrepancy observed between the desired and the actual tension arcs (that is, the quantity E, introduced in Section 4.2). In order to properly evaluate the effect of the method described in this paper, we divided our testers in two groups. Three of our players used the system exactly as described in this paper; the other two used a modified version that doesn't include the mechanism for following a specified tension arc. Comparing the results from the two groups, we can have an idea of the reduction on the discrepancy promoted by our method.

(a) Ugh looking for clues in Brokung's cave.

(b) Rain falling in the forest.

(c) Ugh frightening Brokung.

(d) A new strawberry grows in the garden.

Figure 5: *Some scenes from* Ugh's Story 2.

For the first group, the resulting total discrepancies (E) were 9337.5, 4434.9 and 5972.4 (mean: 6581.6). For the second group, values were 63804.0 and 31251.8 (mean: 47527.9). Although we cannot have any statistical guarantees (since our sample is small), the reduction in the total discrepancy is clear.

6.2 Example Run

Here, we present an excerpt of a story created by a player using our prototype during the experiments. For space and clarity reasons, we include only the events necessary to demonstrate the work of the underlying IS system. This example run is also graphically represented in Figure 6.

03:08 The player executes an action that renders the current plan invalid. So, as described in Section 3, a new plan is generated. The current tension level is slightly above the desired, so the generated plan (using $c_{\text{NPC}} = 103$) doesn't include NPC actions that would help the player:
$GoTo(Ugh, Forest, Patio)$
$TalkAbout(Ugh, Egonk, EgonkHasPersonalStatue)$
$GoTo(Ugh, Patio, AlelughsCave)$
$GivePresent(Ugh, Alelugh, FireStarter)$
$GoTo(Ugh, AlelughsCave, Garden)$
$ExaminePlace(Ugh, Garden)$
$Take(Ugh, DebtCharge, Garden)$
$ExamineThing(Ugh, DebtCharge)$
$GoTo(Ugh, Garden, AlelughsCave)$
$TalkAbout(Ugh, Brokung, BrokungIsIndebted)$

03:40 Rain falls (Figure 5(b)). This is a pretext action being executed by the system.

06:00 For almost three minutes, the player wasn't able to find any new clue. Hence, the tension level got substantially lower than the desired. This situation is detected and a new plan is generated, using $c_{\text{NPC}} = 75$. According to the new plan, shown below, Egonk will go to the garden, examine it to find a new clue, and tell this new clue to the player.
$TalkAbout(Ugh, Egonk, EgonkHasPersonalStatue)$
$GoTo(Ugh, Patio, AlelughsCave)$
$TalkAbout(Ugh, Alelugh, SuspectFireStarter)$
$GoTo(Egonk, Patio, Garden)$
$ExaminePlace(Egonk, Garden)$
$Take(Egonk, DebtCharge, Garden)$
$ExamineThing(Egonk, DebtCharge)$
$GoTo(Egonk, Garden, AlelughsCave)$
$TalkAbout(Egonk, Ugh, BrokungIsIndebted)$
$TalkAbout(Ugh, Brokung, BrokungIsIndebted)$

06:14 As the plan determined, Egonk goes to the garden.

06:15 Egonk examines the garden and finds a hidden document.

06:17 Egonk takes the document.

06:17 Egonk examines the document and notices that it charges Brokung a large debt.

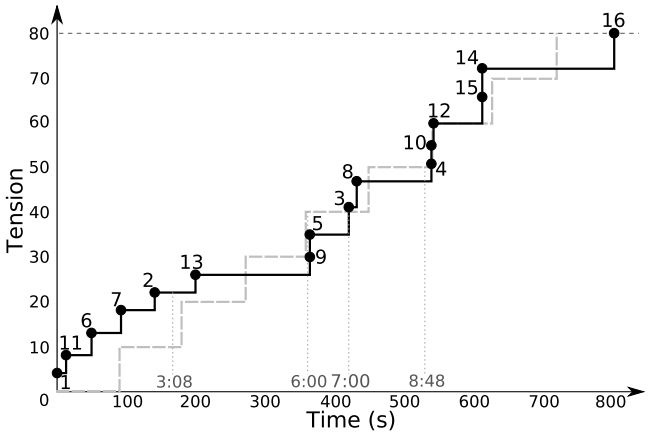


Figure 6: *Tension arc for the story discussed in Section 6.2 (continuous black line). The desired tension arc is shown for reference (dashed gray line). Numbers near the dots refer to the clue that caused the increment in the tension (see Table 1).*

06:40 Egonk goes to Alelugh's cave, where the protagonist is standing.

07:00 Egonk tells the player that Brokung is deeply indebted. With this revelation, the story tension level raises to a level closer to the desired.
At this point, the story is close to the end of its first act. It will advance to its second act if the player interrogates Alelugh and Brokung about some previously found clues. So, the current plan consists of only two actions:
$TalkAbout(Ugh, Brokung, BrokungIsIndebted)$
$TalkAbout(Ugh, Alelugh, SuspectFireStarter)$

08:21 The system detects that the story will reach a dead end if the player executes the following sequence of actions:

1. *Frighten Brokung* (Figure 5(c)). Brokung would get angry with the player, and refuse to talk to him again. (One must recall that it is necessary to talk to Brokung in order to finish the first act).

2. *Give the strawberry (that the player possesses) as a gift to Egonk.* The only way to regain Brokung's friendship would be giving him this strawberry. And, since the only strawberry in the world was given to another character, this would leave the world in an unsolvable state: a dead end.

08:48 With the pretext that the rain at 3:40 irrigated the earth, the system executes the *GrowNewStrawberry* intervention action: a new strawberry grows in the garden (Figure 5(d)). From this point on, the story will no longer enter in a dead end if the player frightens Brokung and gives the strawberry to Egonk, because the new strawberry can be taken and given to Brokung. It worths mentioning that nothing precludes further executions of *GrowNewStrawberry* if more strawberries are needed.

6.3 Observations and Interviews

The players that used the version of the prototype that doesn't include the mechanism to respect tension arcs have faced at least one long interval during which they were not able to find clues. During these periods, they have clearly shown negative feelings, specially boredom and angriness. Players using the complete system have not faced situations like this, and therefore have not shown these feelings.

During the interviews, none of the players mentioned the execution of pretext or intervention actions. We believe that this is a very positive result, since one of our goals was intervening subtly in the stories, in a way that everything looks natural.

7 Final Remarks

The field of Interactive Storytelling is still incipient and deals with a very complex topic. Several different approaches are being used by different research groups. This work is based on the use of a planning algorithm to define the sequence of events that compose the story.

It turns out that planning algorithms are typically created in the context of Artificial Intelligence (AI) research, which has different goals than those in IS. AI is mostly concerned with "hard" and precise goals like optimality. On the other hand, the narrative goals in IS are more subtle, not easily defined formally. Hence, a general purpose planning algorithm can easily find a plan that transforms the world from its initial state to a goal state, but nothing guarantees that this plan will correspond to a good story. Informally, this boils down to "planning algorithms don't know how to tell good stories." Or, in the terminology used in this work, "planning algorithms don't know about narrative consistency."

In this paper, we presented two techniques that can be used in the context of planning-based IS to confront this fact. When the generated stories closely follow an ideal tension arc, the narrative will progress in the pace imagined by its author, transmitting the feelings imagined by its author. Similarly, when an IS system avoids dead ends without being too explicit, it keeps the player immersed in the story, and feeling that his actions are relevant. Our results, obtained from experiments with users, suggest that the proposed mechanisms can help to advance in these directions.

There are certainly many aspects of this work that can be improved in the future. Among these possibilities, we would like to mention the creation of a more complex model of tension arcs for riddle stories. We are particularly interested in dealing with lies told by NPCs, because, when an NPC is belied, the player's knowledge about the riddle will shrink in some sense (and this is not supported by our current model). Another possibility for future investigations concerns the execution of pretext actions. We currently execute them at random moments, but it is possible to think about other heuristics that may offer better results.

Acknowledgments

We would like to thank Marcelo H. Borba for the cavemen models used in *Ugh's Story 2*. This work is supported by the National Council for Scientific and Technological Development (CNPq – Brazil).

References

BARROS, L. M., AND MUSSE, S. R. 2005. Introducing narrative principles into planning-based interactive storytelling. In *Proc. of ACM SIGCHI Intl. Conference on Advances in Computer Entertainment Technology (ACE 2005)*, 35–42.

BARROS, L. M., AND MUSSE, S. R. 2007. Improving narrative consistency in planning-based interactive storytelling. In *Proc. of the Third Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE 07)*.

CHARLES, F., LOZANO, M., MEAD, S. J., BISQUERRA, A. F., AND CAVAZZA, M. 2003. Planning formalisms and authoring in interactive storytelling. In *Proc. of TIDSE'03: Technologies for Interactive Digital Storytelling and Entertainment*.

CRAWFORD, C. 2004. *Chris Crawford on Interactive Storytelling*. New Riders Games, Indianapolis, USA, October.

GLASSNER, A. 2004. *Interactive Storytelling: Techniques for 21st Century Fiction*. AK Peters, March.

HOFFMANN, J. 2003. The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables. *Journal of AI Research, special issue on the 3rd Intl. Planning Competition 20*, 291–341.

LEBOWITZ, M. 1985. Story-telling as planning and learning. *Poetics 14*, 483–502.

MAGERKO, B. 2005. Mediating the tension between plot and interaction. In *Imagina 2005*.

MATEAS, M., AND STERN, A. 2003. Façade: An experiment in building a fully-realized interactive drama. In *Game Developers Conference, Game Design track*.

MURRAY, J. H. 1997. *Hamlet on the Holodeck: The Future of Narrative in Cyberspace*. Free Press, New York, USA.

RIEDL, M. O., SARETTO, C. J., AND YOUNG, R. M. 2003. Managing interaction between users and agents in a multi-agent storytelling environment. In *Proc. of AAMAS '03: the 2nd Intl. Conference on Autonomous Agents and Multi-Agent Systems*.

RUSSELL, S., AND NORVIG, P. 2002. *Artificial Intelligence: A Modern Approach*, second ed. Prentice Hall, Englewood Cliffs, USA, December.

TOBIAS, R. B. 1993. *20 Master Plots: and how to build them*. Writer's Digest Books, Cincinnati, USA.

ZAGALO, N., BARKER, A., AND BRANCO, V. 2004. Story reaction structures to emotion detection. In *Proc. of the SRMC'04, ACM Multimedia 2004 — Workshop on Story Representation, Mechanisam, Context*, 33–38.

Virtual Dungeon Master: Um Narrador Inteligente de *Quests* para *Role Playing Games*

Felipe S. Pedroso João R. Bittencourt

Universidade do Vale do Rio dos Sinos (UNISINOS), Grad.Tec. Jogos Digitais, Brasil

{fspedroso@gmail.com; joaorb@unisinos.br}

Resumo

Este trabalho propõe a criação de um agente inteligente que utiliza técnicas de Inteligência Artificial (Lógica Difusa), capaz de desenvolver e gerenciar *quests* de *Role Playing Game* (*RPG*), incluindo personagens coadjuvantes, monstros e eventos gerais. Será utilizado como base do comportamento do agente o Sistema de Aventuras Instantâneas (SAI), que gera através de jogadas aleatórias, os eventos principais de uma aventura de método RPG (*quest*). Com este método simplificado é possível criar o lugar onde vai começar a aventura, o inimigo principal, o fornecedor das pistas, além de outras características de uma aventura. Para evitar a criação de uma história totalmente inconsistente, devido à aleatoriedade, é adotada uma ontologia que descreve o mundo do jogo e uma base de conhecimentos, modelada com Lógica Difusa que possibilitam ao agente ajustar a narrativa de forma a deixá-la mais imersiva e atrativa em conformidade com o conceito de ficção interativa.

Abstract

This work presents the construction of a intelligent agent that uses Artificial Intelligence (Fuzzy Logic) and is capable to create and manage quests for a Role Playing Game, including non playable characters, monsters and generic events. For the base behavior of the agent it's used the Sistema de Aventuras Instantâneas (SAI), which creates through a random system the main events of an *Role Playing Game* (RPG) campaign. With this simple system is possible to create the place were the adventure begins, the main villain, who gives clues about the plot, besides other features about the adventure. To avoid a incompatible story due the random system, it's created a ontology that describes the world of the game and a knowledge base with Fuzzy Logic rules, that helps the agent to adjust the story to make it more immersive and attractive.

Palavras-Chave: Inteligência Artificial, Role Playing Game, Storytelling.

1. Introdução

Os jogos de *Role Playing Game* (RPG) sempre se destacaram no mercado para computadores e vídeo games. Muitos deles, derivados do sistema papel e caneta, possuindo mecânicas e enredos parecidos com estes. O RPG tem como base as inúmeras *quests* que podem ser criadas, não apresentando limitações quanto às possibilidades de explorar a narrativa. Entende-se por *quests*, missões que necessitam serem cumpridas no decorrer do jogo, tendo em vista o aprimoramento do personagem, seja através da aquisição de novos itens ou da evolução pelo ganho de pontos de experiência. Entretanto, atualmente o enredo segue de forma linear, não deixando espaço para uma maior exploração da narrativa e dos elementos do jogo.

No mercado, existem alguns jogos que possibilitam ao jogador criar suas próprias aventuras. *Neverwinter Nights* [Bioware 2002] é um deles e destaca-se por ter uma solução denominada *Aurora Toolset*, que facilita a criação e edição de *quests*, itens, monstros e *Non-Player Characters* (NPC), utilizando o sistema *Dungeons & Dragons* [Wizards 2000]. Este editor oferece um bom conjunto de possibilidades para montar o mapa de aventura, mas não oferece um meio de criar NPCs de forma automática, pois falas, lugares e aparência precisam ser editados manualmente, o que para muitos RPGistas é um trabalho não muito agradável. Também oferece um editor de *quests*, mas compete ao usuário criar toda a trama.

Murray [Murray 2003] cita a importância da comunicação e interatividade através de narrativas em ambientes informatizados, como por exemplo, jogos e textos digitais. Propõe um novo conceito de narrador, o *Cyberbardo,* que seria um contador de histórias, um típico *Dungeon Master* (papel de um jogador que controla e cria as situações narrativas em uma típica partida de RPG com lápis e papel) podendo alterar a narrativa conforme a mesma se desenrolar. Há possibilidade de criar novas situações e elementos dramáticos, desenvolvendo uma história personalizada e única para o jogador.

Na comunidade científica já existem iniciativas referentes aos algoritmos de *storytelling*, por exemplo, as pesquisas de Cavazza [Cavazza et. al. 2003], onde

foram pesquisados e abordados temas como Inteligência Artificial, usados em sistemas de narrativas, como também o comportamento de cada personagem por meio de árvores de decisões hierárquicas [Cavazz et. al. 2005].

Com estes dados, pretende-se criar uma forma diferenciada de jogar RPGs Computadorizados (*CRPG – Computer Role Playing Game*). A proposta do presente trabalho é desenvolver um protótipo de um agente inteligente autônomo denominado *Virtual Dungeon Master* (VDM) que irá criar *quests* personalizadas ao personagem do jogador. Desta forma, espera-se proporcionar uma melhor experiência ao jogar, adicionando um tempo extra de *replay* ao título no mercado.

Este artigo está organizado em seis seções. Na seção 2 são apresentados os trabalhos relacionados ao tema. Na seção 3 são abordadas questões importantes quanto ao gênero RPG e a ficção interativa. Posteriormente na seção 4 os principais conceitos teóricos utilizados para fundamentar o modelo proposto na seção 5. Na seção 6 são retomados alguns aspectos referentes ao protótipo implementado. Por último, na seção 7 são feitas as considerações finais

2. Trabalhos Relacionados

Cavazza [Cavazza et. al. 2005] propõe um protótipo de *storytelling* interativo. O sistema é focado nos personagens e não no enredo em si. É mostrada uma história e, no seu desenrolar, é possível intervir a qualquer momento, enviando comandos para os atores realizarem. A história é adaptada conforme os atores interagem, essas interações são implementadas a partir de *Hierarchical Task Network* (HTN), um algoritmo que permite a busca através de uma hierarquia de valores definidos.Em outro artigo utilizado como base, Cavazza [Cavazza et. al. 2003] descrevem um sistema chamado OPIATE – *Open-ended Proppian Interactive Adaptive Tale Engine*, que simula o papel do Mestre de Jogo, utilizando técnicas de IA. O sistema controla toda a história apresentada em um jogo de aventura, onde o jogador controla um herói e o sistema reage a cada ação do jogador.

Algoritmos como *Hierarchical Task Network* (HTN) ou *Heuristic Searching Method* (HSM) são usados para resolver este desafio. Fairclough [Fairclough et. al. 2003] apresentam *Fuzzy Cognitive Goal Net* como a ferramenta de criação de enredos, combinando a capacidade de planejamento da *Goal Net* e a habilidade de conclusão da *Fuzzy Cognitive Maps*.

Charles [Charles et. al. 2004] propõem a utilização de Realidade Misturada (*Mixed Reality*) para uma nova maneira de apresentar narrativas interativas. O usuário, além de interagir, irá fazer parte da história como um personagem virtual, desempenhando um papel durante a narrativa. São capturadas, em tempo real, seqüências de gestos que o usuário executa e sobrepõe no mundo virtual, povoado por agentes autônomos que poderão atuar com o usuário.

Liu [Liu et. al. 2006] descrevem o conceito de um banco de histórias visuais interativo (*Visual Storytteling Warehouse* ou VSW) que utiliza *eXtensible Markup Language* (XML) para gerar histórias no estilo *cartoon*, conforme essas foram descritas em uma estrutura , ou interação do usuário. Dada uma estrutura de enredo, o sistema é capaz de produzir múltiplas versões de uma mesma história a partir de personagens, *background* (plano de fundo) e diálogos da história. Estes são extraídos do banco de dados a partir das especificações do usuário.

3. RPG e Ficção Interativa

RPG é a sigla de *Role Playing Game*, ou seja, um jogo de atuação e interpretação de papéis. Foi criado por Gary Gygax e Dave Arneson, em 1974, juntamente com o lançamento do jogo *Dungeon & Dragons*.

Durante uma partida de RPG, os jogadores assumem o papel de um personagem que se aventura em cenários diversos como, por exemplo, fantasia medieval. Ao contrário de um jogo de vídeogame, em uma partida de RPG não há um final pré-estabelecido, os jogadores decidem quando irão finalizar a aventura. Geralmente, é jogado com um pequeno grupo de pessoas, dentre eles, um é selecionado para ser Mestre de Jogo, o narrador, aquele que ditará o andamento e os acontecimentos, ao mesmo tempo controlando *Non Playable Characters*[1] (NPCs) e monstros. Também age como juiz, caso algum empecilho ocorra durante a partida.

Apesar do RPG ser um jogo, não há vencedores ou perdedores, pois o mesmo caracteriza-se por ser cooperativo. Todos ajudam a cumprir os desafios. É prezada a diversão e, conseqüentemente, a criação, a interpretação, o raciocínio, entre outras habilidades sociais.

Jogos de *Computer Role Playing Games* (CRPG), os primeiros a serem feitos para computador, como por exemplo, Ultima[2] [Ultima 80], apesar dos poucos recursos gráficos e de máquinas disponíveis, já apresentavam um princípio de narrativa. Estes foram uma primeira tentativa de transpor o RPG de mesa para um mundo virtual. Tais jogos foram até então muito bem recebidos pelo público. Com a evolução dos computadores foi possível melhorar os CRPGs,

[1] *Non Playble Characters*, no português, Personagens não jogadores. No RPG de mesa são personagens controlados pelo mestre e são interpretados pelo mesmo. Nos CRPGs são controlados por Inteligência Artificial ou lógica do jogo. São personagens com quem o jogador pode interagir, sendo geralmente o ferreiro, o dono da taverna, o guarda local, etc.

[2] Conhecido também como *Akalabeth.*

utilizando-se de recursos de *hardware* mais sofisticados, possibilitando a seus criadores um melhor foco na história a ser desenvolvida.

O gênero CRPG é um dos mais jogados desde o inicio da era de jogos via computador, mas ainda não há uma evolução na parte da narrativa de suas histórias. A linearidade é algo que prevalece até hoje. Não importa o tamanho do mundo que se tem no jogo, pois não é possível explorá-lo livremente e realizar *quests* sem que elas tenham relações dependentes entre si. Isto impede o jogador de avançar livremente, pois só poderá realizar uma determinada *quest* que esteja encadeada linearmente, não havendo possibilidade de avançar sem cumprir a anterior.

Tychsen [Tychsen et. al. 2005] descrevem o papel do DM existente em quatro tipos de plataformas diferentes. A função do DM varia em cada ambiente, *no Pen and Paper Role Playing Game* (PnPRPG) é responsável por todo o desenrolar da história, que em alguns CRPGs não foram substituídos completamente por uma narrativa pré-determinada. Em MMORPGs o conceito é diferente, sendo o GM apenas um observador com poder limitado, não podendo interferir diretamente no ambiente de jogo ou nas regras. Por último DMs em *Live Action Role Playing Games* (LARPs) são forçados a deixar que o jogo tome um rumo aleatório, não tendo controle direto no desenrolar da narrativa.

A narrativa conhecida atualmente vem sendo desenvolvida ao longo do tempo. Os primeiros estudos das estruturas da narrativa foram realizados pelo filósofo grego Aristóteles, através da análise de poemas e epopéias. Ao longo dos anos, alguns autores desenvolveram a fundo a teoria da narrativa, como Quintiliano (escritor e retórico da Roma antiga). Para a teorização na área dos jogos foi escolhida a de Vladimir Propp, que em 1928 publicou *Morphology of the Folktale* [Propp 1983], na qual trata dos elementos narrativos básicos. O desenvolvimento da metodologia e da teoria deu-se através de sua pesquisa com contos folclóricos russos. Propp identificou sete classes de personagens ou agentes, seis estágios de evolução da narrativa e trinta e uma funções narrativas das situações dramáticas.

Vladimir Propp trata das esferas de ação dos contos folclóricos com sete grandes personagens: 1. o Agressor (aquele que faz o mal), 2. o Doador (aquele que entrega o objeto mágico ao Herói), 3. o Auxiliar ou coadjuvante (que ajuda o Herói durante o percurso), 4. a Princesa (ou personagem procurada) e seu Pai, 5. o Mandatário (o que envia o herói em missão), 6. o Herói e 7. o Anti-Herói ou Falso-Herói.

A morfologia que Propp sugeriu (esquematizada abaixo) pode ser aplicada em filmes como *Star Wars*™ ou *Star Trek*™, provando como sua teoria é amplamente usada até os nossos dias.

Nota-se que os jogos digitais em geral, têm sua linha de narrativa inferior ao proposto por Propp, ou seja, apesar de apresentarem o andamento da história de uma forma básica, com alguma possibilidade de interação do jogador, o jogo em si não se adapta ao jogador. Os jogos geralmente seguem uma linha contínua que não pode ser desviada com um único começo e um único final. Alguns trabalham com formas ramificadas de narrativa, mas mesmo com alguns nós de narrativa, não teríamos como tratar todos os finais possíveis. Seriam muito difíceis de programar, pois quanto mais bifurcações, mais complexo torna-se o jogo.

Mudar a narrativa conforme a interação do jogador com o mundo é um dos grandes desafios da programação em jogos, uma vez que *quests*, e a própria história, deveriam adaptar-se conforme o personagem. Ter-se-ia um modelo que renderia um maior *replay*, aumentando a diversão e a capacidade de gerar um mundo novo a cada início de partida.

Logo, o processo de contar estórias é essencial nos jogos de RPG. Entretanto a arte narrativa requer a existência de um narrador apto capaz de criar eventos e adaptar o fluxo narrativo conforme as ações dos demais jogadores. É um senso comum dos jogadores típicos de RPG afirmarem que um "RPG de mesa" (modalidade não computadorizado) proporciona uma experiência muito mais enriquecedora do que as modalidades tecnológicas. O principal diferencia destes dois modos de jogo é a existência de um narrador humano, capaz de julgar e adaptar os eventos para o grupo. No agente inteligente que está sendo proposto neste trabalho espera-se dotá-lo com esta capacitação de adaptação, desta forma criando narrativas mais diversificadas ao invés de utilizar um sistema linear pré-determinado.

Na próxima seção serão abordados alguns conceitos teóricos que permitiram modelar uma heurística de desenvolvimento de estórias.

4. Embasamento Teórico

4.1 Sistemas de Aventuras Instantâneas

Analisando uma partida de RPG, podem-se notar padrões específicos, tanto criados por mestres quanto aleatórios. Baseado nos fatos citados, foi criado o Sistema de Aventuras Instantâneas (SAI) [Filho e Cobbi]. O sistema reúne uma série de opções que podem ser usadas em uma aventura, podendo gerar, se o mestre assim desejar, uma aventura (*quest*) ou campanha completa (seqüências de aventuras).

A proposta de arquitetura que será detalhada na próxima sessão utilizou o SAI, porque esse oferece uma estrutura básica para se criar *quests*. Apesar da sua aleatoriedade, é possível determinar cada aspecto de uma partida de RPG. Foi levada em conta a

praticidade do sistema em gerar aventuras rapidamente, oferecendo facilidade de implementação de um agente que desempenhe o papel de um DM virtual.

O SAI possibilita ao mestre, através de sorteios de dados, selecionar o local em que a *quest* irá iniciar, o contato que será a pessoa que dará a missão ao herói, o objetivo do jogo, dividido em escolta, entrega, coleta, localizar e destruir, caçada e espionagem, o antagonista e seus asseclas e por fim *subquests* que estão espalhadas pelo mundo do jogo, no protótipo aqui descrito, *subquests* são equivalentes a *quests*.

É importante destacar que o SAI não se trata de um sistema computacional, mas um livreto distribuído gratuitamente na Internet para criar estórias para o RPG de mesa.

Destaca-se neste sistema a simplicidade, entretanto pode gerar narrativas sem sentido em função do forte fator aleatório. No caso de uma partida não computadorizada o narrador humano poderá ajustar estas situações não verossímeis para criar um contexto de jogo mais realista. Logo, torna-se necessário dotar o agente narrativo de uma habilidade de efetuar julgamentos, adaptar uma estrutura randômica para uma estruturada mais consistente. Assim, adotou-se a lógica difusa como uma técnica de Inteligência Artificial que permitiria o agente fazer tomadas de decisão.

4.2 Lógica Difusa

Ao desenvolver um sistema inteligente voltado para jogos, pensou-se o quanto seria bom, se o computador pudesse interpretar tais termos como qualquer ser humano. À medida que a IA interpretasse o que fosse sendo pedido, poderíamos apenas interferir no sistema adicionando regras, já que atuaremos como especialistas, adaptando o sistema a nossa vontade.
Processar estes termos com uma lógica convencional seria inadequado. Imaginemos ter de interpretar um jogo de tênis: Se o **vento estiver fraco** e o **campo for de grama** e o adversário for rebater **fraco na rede**, então **corra rapidamente** e **rebata com força** na paralela para marcar o ponto [Buckland 2005]. Estas regras são descritivas para um ser humano, mas são difíceis para um computador interpretar. Palavras como "fraco" ou "forte" não apresentam uma descrição representativa.

A Lógica Difusa foi proposta por Lotfi Zadeh em meados dos anos 60, possibilitando a os computadores processarem termos lingüísticos e regras similares aos humanos. Conceitos como "longe" e "forte" não são mais interpretados por intervalos e sim por conjuntos difusos em que podem ser indicados valores com um determinado grau. A este processo chamamos de fuzzificação. Usando valores difusos, computadores são capazes de interpretar regras lingüísticas e gerar uma saída que seja menos determinística ou mais comum em vídeogames, podendo ser defuzzyficados para um valor *crisp*. O processo da lógica difusa ocorre da seguinte maneira: os valores matemáticos (*crisps*) são transformados em valores *fuzzy*, ou fuzzificação *(fuzzyfication);* são aplicados os operadores *fuzzy*, OR, AND, NOT e a implicação das regras *(fuzzy rules*); por fim todas as combinações das saídas fuzzy possíveis são combinadas e transformadas em um valor matemático novamente *(defuzzification).*

Esta forma difusa de especificar as regras é bastante conveniente quando trata-se de questões narrativas. Dificilmente o especialista deste domínio especificaria alguma regra utilizando uma abordagem booleana clássica. Seu discurso seria repleto de imprecisões, divagações do tipo: *"se o personagem é forte, mas no momento está cansado então ele deveria descansar"*. Como caracterizar o forte? O está cansado? O quanto deverá descansar? Todos estes pontos podem ser descritos por conjuntos nebulosos. Exatamente por esta característica da lógica difusa em tratar com dados lingüísticos e de forma imprecisa que adotou-se tal técnica para modelar o sistema de tomadas de decisão do agente VDM.

4.3 Ontologia

O termo ontologia é originado da filosofia. É o ramo que lida com a natureza e a organização do ser. Diferentes literaturas apresentam definições variadas com pontos de vista distintos. "Uma ontologia é um conjunto de termos ordenados hierarquicamente para descrever um domínio que pode ser usado como um esqueleto para uma base de conhecimentos." [Swartout et. al. 1996]. Na citação já se percebem algumas informações sobre como é a estrutura de uma ontologia. A ontologia deve possuir termos organizados com uma hierarquia associada, ou seja, uma taxonomia. Na descrição, destaca-se uma das principais utilidades da ontologia, servir como base de conhecimentos.

As ontologias compartilham semelhanças estruturais, pois estas geralmente descrevem indivíduos, classes, atributos e relacionamentos. Indivíduos ou instâncias descrevem objetos concretos, por exemplo, pessoas, lugares, animais ou objetos abstratos como números. Classes são grupos abstratos; conjuntos ou coleções de objetos, podendo conter indivíduos, outras classes, ou a junção de ambos. Cada objeto em uma ontologia, pode ser descrito através de atributos, essas informações geralmente são nomes, números, listas específicas para cada objeto. Por último, o relacionamento ou relações entre objetos descreve a semântica do domínio, por exemplo, objeto "A" usa objeto "B", objeto "B" cria o objeto "C".

A popularização das ontologias através da *Internet* gerou o desenvolvimento de diversas bibliotecas. Estas são disponibilizadas através de *sites,* podendo ser livremente atualizadas.

O VDM deve conhecer o contexto de suas tomadas de decisão. Deve saber qual os elementos compõem o mundo de jogo. Assim, descreve-se o conhecimento formal do mundo na forma de uma ontologia que permita o agente efetuar suas tomadas de decisão de forma consistente com o mundo.

5. Proposta da Arquitetura

5.1 Arquitetura geral do VDM

A arquitetura foi planejada para que todos os componentes realizem uma tarefa específica e o VDM possa se utilizar destas, através da troca de informação usando interface pré-definidas. O VDM é o centro da lógica, é ele que irá tomar as decisões que serão apresentadas ao jogador com base nos componentes encontrados no jogo, através de uma relação de dependência. (Figura 1)

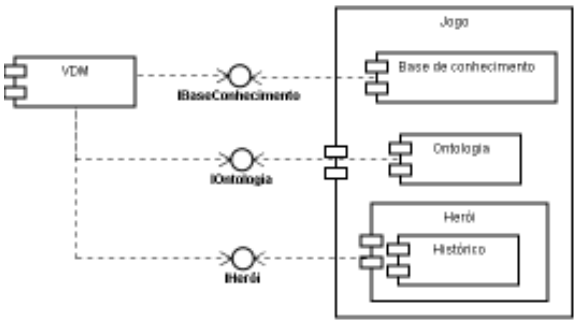


Figura 1: Interação entre o *Virtual Dungeon Master* e o Jogo.

O VDM é encarregado de gerar e gerenciar processos, conforme descrito na seção 5.2. Ele busca informações nos componentes, toda vez que precisa realizar determinada ação como, por exemplo, gerar o mundo de jogo. Quando necessário, pode escrever dados, mas somente no componente Histórico do Herói, esses são atualizados conforme o desenvolvimento do jogo. Assim, consultas mais freqüentes podem ocorrer. Durante o jogo, o VDM deverá aplicar regras para determinadas ações, essas são feitas através da relação com a Base de conhecimento e a Ontologia. Se o VDM fosse totalmente criado como sugerido no sistema SAI, as estórias seriam totalmente aleatórias e possivelmente sem consistência. Para evitar tal problema, associou-se uma base de conhecimentos e uma ontologia para conceder uma espécie de "bom senso artificial" para o VDM.

O componente Jogo é constituído de três componentes descritos a seguir. A Base de conhecimento contém a priori um conjunto de regras. Poderão ser adotadas diferentes técnicas de IA para implementar tais regras, desde regras de predições simples até uma base de conhecimento difusa. As regras são baseadas nos atributos do Herói e seu estado atual, sendo eles nível, quantidade de pontos de vida, raça e língua falada. Gerando diversas saídas de dados como *quests* possíveis de serem feitas pelo personagem, e sugestões de recompensas (será detalhado na seção 5.2).

A Ontologia é responsável por armazenar a descrição do mundo de jogo, contendo seus terrenos, locais, itens, monstros, classes de personagens, entre outras informações. O VDM apenas poderá ler o conteúdo armazenado para poder montar o mundo como está descrito, não podendo escrever dados no componente Ontologia.

O último componente do jogo é o Herói. Esse é o personagem principal, já que todo o jogo se desenvolve a partir do mesmo. Este contém diversos atributos conforme a regra do sistema de RPG adotado. Ao contrário dos demais componentes, o VDM poderá incluir dados ou mudar os já existentes. Sendo sorteada uma *quest* para o Herói, a descrição e demais dados são inclusos no componente histórico, também são armazenadas informações a respeito da quantidade de vezes em que *quests* e locais foram sorteados. Quando o jogador realiza um *quest* com sucesso, o VDM informa o Histórico do acontecido, mantendo assim uma espécie de *ranking*, ou seja, se uma *quest* de escolta for dada ao Herói duas vezes seguidas, a próxima não poderá ser do mesmo tipo, assim mantendo sempre uma alternância, evitando a repetição demasiada. O mesmo acontece para os locais já visitados pelo Herói.

É importante destacar que o VDM foi desenvolvido para que possa ser usado independente da implementação do jogo. A comunicação é estabelecida através de interfaces. Caso seja necessário trocar a temática e regras, poderá ser feito apenas adaptando as interfaces conforme o desejado. Isto torna o VDM um sistema versátil, uma vez que não está limitado a apenas um tipo de jogo. Por exemplo, esta arquitetura do VDM poderá ser usada para criar um jogo de RPG em 3D, basta que o título implemente uma ontologia, uma base de conhecimentos e um herói. O VDM vai utilizar estas informações para criar uma quest que vai ser adicionada no histórico do Herói. A maneira como o jogo vai tratar esta quest, seja através de descrições textuais, colocadas no console, ou através de modelos tridimensionais está fora do escopo do VDM.

5.2 Processos para Criar *Quests* adotados pelo VDM

A arquitetura de processos (ou ações) que o *Virtual Dungeon Master* executa será descrita detalhadamente nesta seção. A criação da *quest* é feita através de processos, consultas a objetos e tomadas de decisões, em que o jogador poderá decidir a respeito do caminho a ser tomado.

O sistema trabalha com quatro comandos base:

a) Inicializar: este processo ocorre quando o jogo é carregado pela primeira vez, com isto são carregadas as informações necessárias no decorrer da execução do programa. Esta ação é executada somente no início para carregar o mundo e o herói;

b) Sortear: processo usado quando for necessária uma ação aleatória por parte do VDM. Estas ações aleatórias são baseadas nas tabelas do SAI;

c) Escolher: quando for necessária uma interação do jogador, este processo é ativado. A partir da escolha, é possível mudar o curso do jogo;

d) Definir: a partir da escolha do jogador, o VDM pode definir determinadas ações, geralmente dependentes dos objetos a serem consultados. Estas definições são feitas, utilizando uma base de conhecimento.

O primeiro passo a ser executado é inicializar o mundo de jogo. Este está descrito na Ontologia de cada jogo. O jogo existente é composto de um mundo povoado por NPCs, monstros, localidades e itens. Esses são lidos e armazenados pelo VDM para serem usados na medida em que se tornarem necessários.

Na seqüência, é necessário inicializar o herói que representará o jogador. Este é escolhido pelo jogador dentre as diversas possibilidades apresentadas no jogo. O objeto herói é inicializado contendo seus atributos. É criado um histórico vazio.

A primeira tomada de decisão ocorre após a inicialização do herói (diamante A, Figura 2 - Anexo). Na primeira vez em que o jogo for inicializado, é sorteado um terreno dentre todos os possíveis descritos no mundo. Esta ação foi planejada para que a cada vez que se começa um novo jogo, este não ocorra sempre na mesma localidade, possibilitando ao jogador uma nova experiência a cada partida. Se não for a primeira vez, cabe ao jogador escolher para qual terreno mover o herói. Estando em um terreno, é possível escolher entrar em seu agrupamento ou movimentar-se para outro. Estando no agrupamento, duas novas opções são oferecidas (diamante B, Figura 3), pode-se voltar para o terreno inicial ou avançar para os locais que pertencem ao agrupamento. Cada agrupamento contém um número pré-determinado de locais, dependendo da cidade que o jogador escolher. Esta poderá ter ou não certas construções, como por exemplo, um templo ou castelo. Escolhido o local (diamante C, Figura 3), o VDM consulta a base de conhecimentos e o atributo pontos de vida do herói para verificar se o local desejado contém ou não, uma *quest*. Caso não seja constatada uma regra válida para a geração do local, este estará vazio, não apresentando interação, obrigando o jogador a voltar e re-selecioná-lo. Como o local depende dos pontos de vida do herói, há possibilidade de que locais previamente visitados possam estar povoados.

Estando o local de acordo com as regras, este então receberá um Mandatário [Propp 83]. As regras da base de conhecimento são verificadas para escolher qual melhor NPC. Por exemplo, podem existir regras na base de conhecimentos que são baseadas na raça e na língua falada pelo herói. Sempre será gerado um Mandatário, mas com raças e comportamentos diferentes. Um humano terá grandes chances de que seu mandatário seja da mesma raça, mas se este falar élfico, poderá ser apresentado um elfo ou meio-elfo. Este procedimento foi adotado para novamente diversificar o jogo.

O Mandatário dará ao jogador uma missão a ser cumprida. A formação da missão considera regras existentes e o atributo nível do herói. Seis tipos de missões são possíveis, somente uma poderá ser sorteada, ou seja, o mandatário poderá requerer uma *quest* de escolta, mas nunca uma escolta e entrega ao mesmo tempo. Cada classe tem prioridades em suas missões, um mago realizará entregas e coletas mais freqüentes e um guerreiro, caçadas e escoltas.

Determinado o tipo de *quest* que será atribuída ao herói, algumas definições adicionais são feitas. As descrições de cada tipo serão exemplificadas a seguir:

a) Escolta: o VDM define o local em que será concluída a *quest final* e um NPC (chamado de Popular na Ontologia). Para cumprir a missão será necessário levar este NPC para o local determinado.

b) Espionagem: a cidade que deverá ser espionada é definida dentre qualquer uma existente no mundo (menos a cidade em que foi adquirida a missão). Para cumpri-la, é preciso entrar no agrupamento definido.

c) Entrega: definido um item qualquer, será preciso entregá-lo a um determinado local (templo, taverna, guilda,...), em uma determinada cidade.

d) Coleta: o objetivo dessa missão é coletar um determinado item e entregar para o mandatário que o pediu.

e) Procura e destrói: definido um anti-herói de acordo com as regras necessárias, o objetivo é derrotar este inimigo.

f) Caçada: parecido com a *quest* anterior, mas é definido um monstro errante qualquer que pertença ao terreno atual em que o herói se encontra.

Dada a *quest*, esta é armazenada no histórico. O histórico mantém o VDM atualizado sobre as missões completas e pendentes que o herói possui. Ao mesmo tempo, o histórico monitora a quantidade de vezes que determinado tipo de *quest* foi dada ao herói, o local em que o herói recebeu e o tipo de item dado na recompensa. Quanto mais for sorteado um elemento, menos chance este tem de ser escolhido novamente. Assim, existirá uma grande variação dos tipos de *quests*, locais e itens.

Durante a transição de um terreno para o outro, é sorteado um monstro de acordo com o nível atual do herói. Caso o personagem morra, (diamante D, Figura 3) o jogo é terminado e será necessário reiniciá-lo.

Desta forma, as quests são criadas em tempo real em conformidade com o herói e as ações do jogador. Assim, evita-se que as *quests* sejam pré-determinadas. A principal vantagem disso é possibilitar ao jogador experiências de jogo inéditas.

6. Metodologia do Projeto

6.1 Utilização do Sistema d20

Na descrição do mundo, foi usado o cenário de fantasia medieval *Forgotten Realms*™ criado por Ed Greenwood e publicado pela *Wizards of the Coast* em 1990 [Wizards 2000]. Apenas alguns nomes de localidades foram usados, com finalidade de exemplificar o mundo já existente.

Para a construção do jogo, foi adotado o sistema d20 [Wizards 2000], por ser de licença aberta e possibilitar as modificações de acordo com o estilo de jogo do grupo. O d20 oferece regras detalhadas de construção de personagens, monstros e itens, regras para ações diversas, tais como atacar um inimigo desprevenido, deslocamento, magias e batalhas. Na implementação foram seguidas algumas regras que o sistema oferece, sendo elas:

a) *Construção de personagem*: foi seguida de acordo com o descrito pelo sistema. O personagem é constituído por vinte e quatro atributos como, por exemplo, força, destreza, inteligência.

b) *Construção de inimigos*: semelhante à construção do personagem, com algumas modificações referentes ao nível, substituído por nível de dificuldade (ND) que equivale ao nível do monstro em relação ao do herói e adição de atributos como organização e tipo.

c) *Construção de item*: para a construção da ontologia, as regras de construção de itens foram simplificadas. O item é constituído por quatro atributos: nome, tipo, classe (quem pode usar) e preço. Ao contrário do sistema d20, na implementação do VDM, não foram relevadas todas as regras, pois os itens somente serão usados em duas *quests*, não sendo relativamente importantes na implementação atual.

c) *Batalha*: o sistema de batalha foi implementado no VDM como descrito no livro de regras do d20. Primeiro, é verificada a iniciativa (entre o herói e monstro/anti-herói/assecla), aquele com maior iniciativa ataca primeiro. A batalha acontece com alternância de turnos. Para conseguir acertar o oponente, é preciso realizar um ataque com sucesso. De acordo com a regra do sistema, é sorteado um dado de 20 faces. O resultado é somado ao ataque base (baseAtaq) do atacante. Se o número for maior ou igual à classe de armadura (CA), é acertado o golpe.

Caso o ataque seja bem sucedido, é subtraído o dano dos pontos de vida do oponente. Aquele que chegar a zero ou com menos pontos de vida é declarado morto e a batalha é encerrada.

6.2 Descrição da Ontologia

Optou-se pela criação de uma ontologia para descrever o domínio da aplicação. O sistema apresentado na seção 4.1 é usado como base para a formação da estrutura de hierarquia do XML, que por sua vez consiste da ontologia propriamente dita. Foi levada em consideração a praticidade do SAI na criação de enredos, a partir de tabelas com valores pré-estabelecidos. O mundo de jogo foi estabelecido a partir de cada elemento pertencente ao sistema SAI.

A hierarquia foi construída para que certos campos sejam encadeados, ou seja, um terreno sempre terá pelo menos um agrupamento e esse, pelo menos um local. O terreno apresenta ainda diversos monstros errantes (estes distribuídos conforme região encontrada). A última relação de dependência é do anti-herói com seus asseclas (Figura 3).

Para a construção da Ontologia foi utilizada a linguagem de descrição XML, pois facilita na criação de documentos organizados de forma hierárquica, como o banco de dados.

Outras funcionalidades foram levadas em conta para escolha do XML: simplicidade da linguagem e sua formatação, possibilidade de criar *tags* sem ter um limite estipulado, integração com outros bancos de dados, hierarquia de fácil acesso e visualização em diferentes *browsers*.

Tendo o mundo previamente criado, foi utilizado o programa *Protégé* [Protégé 2007] para a criação do XML automático. Com o *Protégé* o processo de adição dos dados no arquivo de XML foi facilitado, pois é possível criar tanto a hierarquia de *tags* quanto as instâncias de conteúdo.

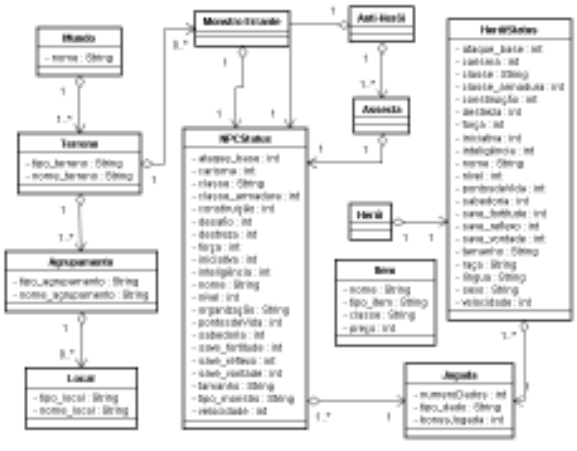


Figura 3: Hierarquia base da Ontologia.

6.3 Utilização do FuzzyF e a Criação das Regras

Para a construção da base de conhecimentos foi utilizado o sistema chamado FuzzyF [Bittencourt 2002], implementado a partir de uma licença livre e que possibilitou a integração com o VDM. O sistema trata da lógica difusa, descrita na sessão 4, que é o ponto vital da base de regras.

Para cada classe existente no sistema d20 foi desenvolvido um conjunto de regras possível de ser aplicado a cada uma. O VDM manda informações referentes a atributos encontrados na descrição do herói, onde serão interpretadas de acordo com a necessidade de cada uma. As informações são enviadas via valores numéricos (*double*). Um conjunto de regras é constituído de entrada (input) (Tabela 1) e saída (output) (Tabela 2). Na entrada, é necessário o fornecimento de informações tais como: nível do herói, pontos de vida, raça, língua e a classe do herói, esta é o próprio conjunto de regras. Cada informação será validada pela lógica do FuzzyF. Na saída, são geradas as informações a respeito do local específico, anti-herói, mandatário, tipo de *quest* e itens. (Anexo A)

Tabela 1 – Descrição das variáveis de entrada

	Entradas
Variável	Descrição
Pontos de Vida	quantidade de vida atual do Herói
Classe	profissão do Herói. Ex: Guerreiro
Raça	etnia do Herói. Ex: Elfo
Língua	línguas que o Herói pode falar
Nível	classificação de 1 a 20

Tabela 2 – Descrição das variáveis de saída

	Saídas
Variável	Descrição
Locais	onde se pode adquirir uma quest.
Desafio	nível referente ao Anti-Herói
Quests	missões que o Herói pode cumprir
Itens	objetos acordados nas regras

Pelo sistema FuzzyF, a validação é feita através das entradas. Cada regra específica, como por exemplo, o local onde a quest será dada ao herói, requer uma entrada a fim de ser validada (Figura 4). O FuzzyF realiza a inferência de pertinência e apresenta os valores de saída.

```
#Regra para locais se ranger estiver com HP
Alto
BLOCK_RULES
RL = IF PV IS Alto THEN TavernaEsc IS Alta
RL = IF PV IS Alto THEN GuildaEsc IS Baixa
```

Figura 4 – Regras referentes à escolha do Local.

6.4 Estudo de Caso

Após toda teorização, partiu-se para a criação de um protótipo de jogo digital onde pudesse ser testado o VDM em relação a sua aplicabilidade. Utilizou-se a linguagem de programação JAVA e uma interação base para com o jogador, via teclado.

Ao iniciar o jogo, é solicitado ao jogador que escolha um herói cadastrado na ontologia. O VDM sorteará um terreno-base no qual o herói iniciará o jogo. Toda vez que o herói se movimentar pelo terreno, ele será atacado por um monstro errante (sorteado de acordo com o nível do herói). Dá-se início a uma batalha automática, ou seja, sem a interferência do jogador. Se o herói vencer a batalha, serão mostradas opções de movimentação. É possível movimentar-se entre os terrenos existentes ou entrar no agrupamento. Neste, duas são as opções possíveis: retornar ao terreno inicial ou fazer uma verificação dos locais existentes. São mostrados todos os locais possíveis de serem visitados (taverna, castelo,...). O jogador tem liberdade para escolher o local onde deseja entrar. Uma vez dentro do local, o VDM verifica a existência do mandatário conforme os status necessários ao personagem. Caso o mandatário for gerado corretamente, este dará ao herói uma *quest* dentre as seis disponíveis (descritas na sessão 5.2). Cabe ao jogador, o cumprimento da *quest* que lhe foi dada.

Uma vez que a missão foi cumprida com sucesso, o jogador deverá retornar ao mandatário com a finalidade de receber sua recompensa. O mandatário poderá, ou não, entregar-lhe a recompensa.

A qualquer momento, o jogador poderá pressionar a tecla "Q" para verificar todas as *quests* pendentes e/ou já cumpridas pelo seu personagem.

Se por acaso, o herói vier a morrer no desenvolver do jogo, o mesmo é encerrado.

Interagindo várias vezes com o programa, são gerados eventos diferentes, inclusive o início do jogo sempre é um lugar diferenciado. Conforme o personagem escolhido, troca-se os tipos de quests, no caso de escolher um guerreiro, é dada preferência por missões bélicas e no caso de escolher um personagem ladino são apresentadas mais *quests* de espionagem.

O processo de integração com VDM foi bastante simples em função da criação de uma arquitetura flexível. Ficou exemplificado um caso de uso utilizando-se dos recursos do VDM com interação da base de conhecimento e da ontologia. O programa mostrou-se válido, gerando as *quests* e demais eventos referentes ao personagem escolhido pelo jogador, em função de gerar estórias consistentes e diferenciadas conforme o personagem e os lances do jogador.

7. Conclusão

Este trabalho foi desenvolvido a partir da idéia de criar um sistema que construísse estórias automáticas de maneira inteligente. Seu principal objetivo foi implementar um sistema que utilizasse técnicas de Inteligência Artificial, capaz de gerar um comportamento consistente quanto à criação de estórias. No desenvolvimento do trabalho, foi proposto o VDM, buscando criar um ambiente controlado, que possibilite ao jogador ter um mundo com variadas alternativas de *quest*, visando com isto a aumentar o interesse, a expectativa e o *replay* dos envolvidos, ao mesmo tempo em que possa dar uma longevidade à vida útil dos jogos, tornando-os mais arrojados e atrativos.

Foi apresentada uma teorização da IA direcionada aos jogos digitais. Descreveu-se amplamente a teoria da narrativa como suporte na construção do modelo aplicado.

A criação deste novo modelo teve como base o Sistema de Aventuras Instantâneas que, apesar de aleatório, forneceu subsídios para a criação do modelo, dando-lhe maior nível de controle.

Espera-se com este trabalho ter contribuído para a aplicação do VDM possibilitando adaptar-se à diversidade de jogos digitais no gênero RPG encontrados no mercado, pois independente da temática dos jogos e suas regras, o VDM é de possível aplicabilidade. Apesar de no trabalho estar sendo utilizada a lógica difusa, é possível trocar por qualquer outra técnica de IA, pois o VDM tem a flexibilidade de ser implementado conforme as regras definidas pelo desenvolvedor da base do jogo. O modelo do VDM foi proposto para ser independente de especificidades de implementações.

A versão inicial implementada do VDM ainda carece de um maior tempo de estudo e aprofundamento, pois sua complexidade exige uma adequação à indústria de jogos, visando a sua larga aplicação no mercado.

Uma nova fase do trabalho deverá estar focada no desenvolvimento do motor, incluindo maior número de regras conforme o sistema adotado. Além destas, destaca-se a necessidade de um visualizador do ambiente gerado pelo VDM bem como a melhoria da interface com o jogador.

Um maior tempo dedicado à construção de uma narrativa, incluindo um perfil de cada usuário, sendo possível uma adaptação da história conforme o próprio perfil e ações tomadas no decorrer do jogo. Também é necessária uma maior adaptação do sistema d20 como distribuição de pontos de experiência a cada monstro derrotado e *quests* cumpridas, o que implicaria na evolução do personagem. Magias e ataques especiais bem como suas regras específicas deverão ser implementadas, para melhor caracterizar o sistema.

Com relação à ontologia e à utilização do XML, deve-se procurar uma generalização das regras e suas variáveis, fazendo com que estas sejam de fácil adaptação, conforme forem necessárias em outros sistemas e modos de jogo.

Através do estudo das diferentes temáticas foi ampliada a base de conhecimentos teóricos, bem como a oportunidade de melhoria da conceituação e validação das teorias que deram suporte técnico ao desenvolvimento do presente trabalho.

Pretendeu-se de maneira prática, buscar uma maior atratividade para o mercado de jogos, visando a atender aos interesses de seus jogadores, proporcionando-lhes múltiplas alternativas de fidelização do cliente e conseqüentemente maior rentabilidade do desenvolvedor.

Referências

BIOWARE CORP. 2002. Disponível em: www.nwn.bioware.com. [Acesso em 20 jun. 2007]. CD-ROM, Windows.

BITTENCOURT, J. R.; OSÓRIO, F. S. FuzzyF - Fuzzy Logic Framework: Uma Solução Software Livre para o Desenvolvimento, Ensino e Pesquisa de Aplicações de Inteligência Artificial Multiplataforma. *In: III Workshop sobre Software Livre, Porto Alegre: Sociedade Brasileira de Computação, 2002*. 58-61.

BUCKLAND, M. *Programing Game AI by Example. Wordware Publishing Inc: Texas, 2005*. 495p.

CAVAZZA, M; CHARLES, F.; MEAD, S. J. *Interactive Storytelling: From AI experiment to new media. In*: Proceedings of the second international conference on Entertainment computing, *May 08 - 10, 2003* , Pittsburgh, Pennsylvania.

CAVAZZA, M; CHARLES, F.; MEAD, S. J. *Interacting with Virtual Characters in Interactive Storytelling. In: Proc. AAMAS'02, July 15-19,* 2005, *pages* 318-325, Bologna, Italy.

CHARLES, F; CAVAZZA, M.; MEAD, S. J.; MARTIN, O.; NANDI, A.; MARICHAL, X. *Compelling Experiences in Mixed Reality Interactive Storytelling. In: Proc. ACE'04*, *June* 3-5, 2004, *pages* 32-41, Singapure.

FILHO, V.; COBBI, B. SAI - *Sistema de Aventuras Instantâneas. Royal Hunter. 2002. Disponível em:* www.rederpg.com.br/portal/modules/news/article.php?storyid=573. [Acesso em 22 jan. 2007].

FAIRCLOUGH, C. R.; CUNNINGHAN, P. *AI Structuralist Storytellingin Computer Games.*

LIU, Y.; AKLEMAN, E.; CHEN, J. *Visual Storytelling Warehouse*. 2006.

MURRAY, J. H. *Hamlet* no *Holodeck: O Futuro da Narrativa No Ciberespaço*. Tradução de Elissa Khoury Daher e Marcelo Fernandez Cuzziol. São Paulo: UNESP, 2003. 282p.

PROPP, V. *Morfologia do Conto*. Tradução de Jaime Ferreira e Vitor Oliveira. Lisboa: Vega, 1983. 2ª Edição. 286 p.

PROTÉGÈ. Stanford Medical Informatics. 2007. www.protege.stanford.edu. [Acesso em 03 de jun. 2007]

SWARTOUT, B.; PATIL, R.; KNIGHT, K.; RUSS, T. *Toward Distibuted Use of Large-Scale Ontologies*. *September* 27, 1996.

TYCHSEN, A.; HITCHENS, M.; BROLUND, T.; KAVAKLI, M. *The Game Master. In:* Proceedings of the *Second Australasian Conference on Interactive Entertainment*, *November* 2005, *pages* 215-222, Sydney, Australia.

ULTIMA. Origin Systems, Inc., 1980, Floppy disk, Apple II, Atari, Commodore 64, DOS, FM Towns, MSX.

WIZARDS OF THE COAST INC., a subsidiary of Hasbro, Inc., 1995 – 2007.

Anexo – Figura 3

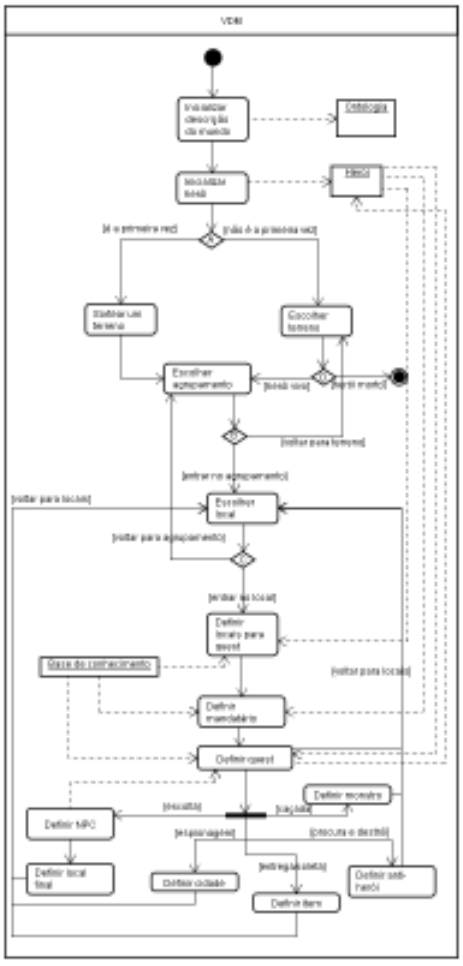


Figura 3: Processos do *Virtual Dungeon Master*.

Anexo – Log da execução do jogo com o personagem Drizzt (Ranger)

```
Bem-vindo jogador ao mundo de Faerûn.
Jogador, escolha o Herói para começar a
partida:
0 : Breyarg Stonebreaker, Classe : Paladino,
Nível: 6
1 : Drizzt, Classe : Ranger, Nível: 16
2 : Sirion, Classe : Mago, Nível: 1
1
Você se encontra em Floresta Neverwinter
0: Deserto de Anauroch
1: Ilhas Moonshae
2: Floresta Mística
3: A espinha do mundo
4: Vale do Vento Gélido
5: Evermoors
6: Planícies Brilhantes
7: Floresta Neverwinter
8: Charcos Prateados
9: A grande floresta
10: Ir para agrupamento
10
Você se encontra em Neverwinter.
O que deseja fazer?
0: Voltar para Terreno
1: Ir para Locais
1
(Fuzzy aplicada quando o herói entra na
cidade, conforme seu hp, será gerado o local
para a quest).
Locais que podem ser visitados:
0 : Armeiro
1 : Casa povoada
2 : Castelo
3 : Ferreiro
4 : Forte Imperial
5 : Grande Biblioteca
6 : Laboratório
7 : Guilda de Mercadores
8 : Porto de Neverwinter
9 : Praça central
10 : Templo de Helm
11 : The Sunken Flagon
12: Voltar para Agrupamento
4
Você se encontra em Forte Imperial.
Você entra no local e não encontra ninguém.
Decida para onde ir:
Locais que podem ser visitados:
0 : Armeiro
1 : Casa povoada
2 : Castelo
3 : Ferreiro
4 : Forte Imperial
5 : Grande Biblioteca
6 : Laboratório
7 : Guilda de Mercadores
8 : Porto de Neverwinter
9 : Praça central
10 : Templo de Helm
11 : The Sunken Flagon
12: Voltar para Agrupamento
10
Você se encontra no Templo de Helm.
Não sei quem é você estranho, mas se quiser
ajudar-me tenho uma proposta a lhe fazer.
Aventureiro, cabe a você derrotar este temível
vilão. Vá e me traga a cabeça de Harpia.
```

Após ter adquirido a quest o jogador poderá então percorrer o mundo para realizá-la, quando cumprida ao retornar para o mesmo local, o mandatário irá dar a recompensa ao herói.

Simulação Visual de Ondas Oceânicas em Tempo Real Usando a GPU

Alex Salgado Aura Conci Esteban Clua

Universidade Federal Fluminense, Niterói, Brasil

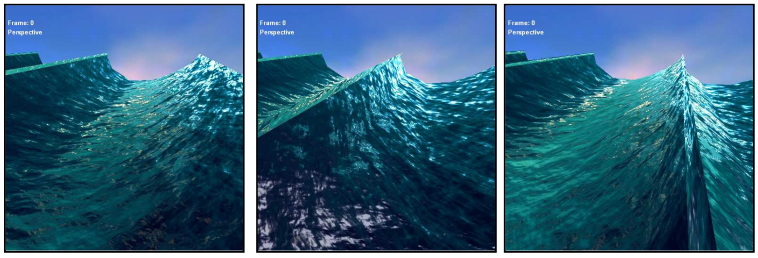


Figura 1: Simulação de ondas em tempo real usando equações oceanográficas

Abstract

Modeling natural phenomena is one of the most challenging tasks in computer graphic. The ocean wave simulation is included in this challenge. With the GPU graphical processing, it is possible to use advanced rendering techniques with great realism in real-time applications. This work simulates the ocean wave behavior processing all geometric computation and rendering in GPU. The shape is defined by Gerstner´s equation where the water particles movement is simulated. It is possible to consider both the representations: deep water and shallow water. The method also simulates the breaking waves near the shore and the wave refraction, which is the topology deep sea influence on its movement.. The implementation of this work uses combinations of advanced real-time rendering techniques of tangent-space reflective bump mapping and environment mapping to Fresnel reflection and HDR. As illustrated in session 6, this technique creates a very realistic waves animation and adjusted scenario.

Keywords: GPU, real-time rendering, ocean waves, water, breaking waves, natural phenomena, vertex and fragment programming

Authors' contact:
{asalgado,aconci,esteban}@ic.uff.br

1. Introdução

Elementos da natureza (terrenos, nuvens, água, árvores, etc) são objetos complexos de serem modelados devido à sua natureza fractal de geometria. Por outro lado, são de fundamental importância na criação de cenários virtuais, tanto para jogos com para aplicações de simulação e visualização. A natureza por si só já é complexa, mas hoje é possível gerar imagens impressionantes, graças à modelagem de fenômenos naturais utilizando alguns modelos físicos e estatísticos. Entre esses, pode-se citar a simulação de líquidos, fogo e gás (Fedkiw, Stam e Jensen, 2001)(Foster e Fedkiw, 2001). No entanto, criar e renderizar águas é uma das tarefas mais onerosas da computação gráfica. Uma renderização realística da água requer que a incidência da luz do sol e a iluminação do céu estejam corretas, que a modelagem da superfície da água esteja real e que o transporte de luz no corpo da água e o seu movimento seja captado corretamente.

Até pouco tempo os jogos tratavam a água como uma superfície plana, limitando-se a aplicar uma textura de modo a assemelhar-se com a cor da água, não sendo consideradas as suas propriedades físicas.

A maioria dos trabalhos considera a modelagem geométrica das ondas através de duas formas de modelagem: física ou empírica. No modelo físico, adota-se a equação de Navier-Stokes, que é baseada na dinâmica dos fluidos para representar o movimento da água através do tempo (Kass e Miller, 1990) (Foster e Metaxas, 1996) (Chen e Lobo, 1995). Devido a seu grande custo de processamento, este modelo ainda é inviável para processamento em tempo real. A abordagem com base totalmente física requerida para estudos científicos é bem diferente da abordagem para jogos em termos de precisão e fidelidade dos cálculos. A outra forma de modelagem é através dos modelos empíricos, os mais conhecidos são de Fournier e Reeves (1986), Peachey (1986), Ts´o e Barsky (1987), Imamiya e Zhang (1995). Estes modelos são baseados no modelo clássico de ondas de Gerstner (Kinsman, 1965) em que a busca para a representação da fidelidade visual é maior do que a fidelidade aos fenômenos físicos.

O objetivo deste trabalho é modelar e visualizar a superfície das águas do oceano em tempo real na GPU, representando inclusive efeitos complexos como a influência do vento, do relevo do fundo do mar e o efeito de ondas se quebrando. Para isto será desenvolvido um modelo de representação da geometria e um sistema de visualização de ondas, através da aproximação empírica. Este sistema poderá

ser usado em jogos, simuladores ou em aplicações que necessitem representar oceanos em tempo real.

Neste artigo será utilizado a GPU (Graphic Processing Unit) para o processamento dos vértices e dos fragmentos. A equação da superfície trocóide (uma generalização de uma ciclóide) é usada como base para a modelagem da forma da onda de acordo com o trabalho realizado por Fournier e Reeves (1986). A animação e a renderização é feita utilizando um modelo de sombreamento (shader) programado na linguagem Cg da NVIDIA, sendo aplicadas técnicas de reflexão por bump mapping no espaço da tangente, environment mapping, reflexão de Fresnel (Wloka, 2002) e HDR (High Dynamic Range).

2. Trabalhos relacionados

Tem havido grande evolução das técnicas de computação gráfica na modelagem realística da água, bem como na sua visualização e animação (Iglesias, 2004). A água, por ser um fluído, pode mover-se em direções e caminhos complexos. A variável "tempo" deve ser incluída nas equações para garantir o movimento do fluído e movimentação adequada.

O termo realismo possui diferentes significados na computação gráfica e depende exclusivamente do objetivo a que se deseja alcançar. Enquanto numa aplicação de visualização científica o modelo de iluminação de cena não é a prioridade, mas sim a precisão dos comportamentos, num jogo espera-se uma iluminação real, mas não uma física precisa. Tendo isto em conta alguns trabalhos recentes abordam diferentes níveis de realismo na simulação de águas (realismo físico e foto-realismo) (Adabala e Manobar, 2002) (Ferwwerda, 1999).

A primeira tentativa de visualizar ondas da água utilizou a técnica de bump mapping (Blinn, 1978). Esta técnica também é usada neste trabalho. Este método permite obter superfícies com rugosidades realistas através da perturbação da normal da superfície modelada, mas não permite a criação de ondas grandes.

Fishman e Schachter (1980) introduziram a técnica de "height field" e posteriormente Max (1981) desenvolveu outro método visando o realismo na simulação do oceano, principalmente a visualização da textura do mar quando a câmera está próxima da superfície. Seu modelo hidrodinâmico foi baseado na idéia de que um modelo de onda é representado por uma solução aproximada (válido apenas para ondas de pequenas amplitudes) em que a velocidade da onda v é proporcional a raiz quadrada do comprimento de onda L.

$$v = \sqrt{\frac{gL}{2\pi}} = \sqrt{\frac{g}{k}} \qquad \text{(Eq 1)}$$

onde

$k=2\pi/L$ - é chamado o número da onda (o espacial análogo da freqüência) e g é a força da gravidade.

Max também usou a primeira aproximação linear do modelo de Stokes (uma série de Fourier infinita a qual se assemelha as ondas trocoidas quando a série é usada com termos além das de terceira ordem).

Entre 1986 e 1988, com a preocupação da interação entre sólido e líquido, foram desenvolvidos técnicas que simulam refração e colisões. Dentre estes trabalhos, destaca-se o sistema de partículas de Reeves (1983) usados na simulação de espumas nas ondas. Peachey (1986) considerou que a superfície pudesse ser modelada usando "height field" e também conseguiu grande realismo em seu trabalho. Fournier e Reeves (1986) é um clássico sempre referenciado nos trabalhos atuais, baseado no modelo de Gerstner-Rankine proposto na oceanografia séculos atrás (Gerstner, 1809).

No modelo de Fournier-Reeves, os autores assumem que a partícula da água descreve uma órbita estacionária circular ou elíptica. Este é o modelo geométrico adotado neste trabalho, como será discutido na seção 4). Formas de ondas realistas e outros efeitos necessários, tais como os relacionados à profundidade (refração e quebras) e o vento, podem ser reproduzidos variando-se alguns parâmetros da equação da órbita deste modelo. Para controlar o formato do oceano Founier e Reeves (1986) introduziram alguns "trens de ondas", i.e. grupos de ondas compartilhando as mesmas características básicas (altura, período e amplitude de onda) e a mesma fase original. No trabalho Founier e Reeves (1986) também foram gerados efeitos de spray e espuma.

Até o final de 1988, alguns fenômenos físicos relacionados à água que ainda não havia sido explorados, começaram a ser considerados. A maior questão era a descrição exata da dinâmica de fluídos.
Por outro lado, alguns efeitos interessantes não haviam sido considerados até então, tais como: simulação de ondas refletidas, a interação entre luz e água, a análise caótica, etc. Para superar essas limitações, vários novos modelos foram criados para simular a dinâmica de fluídos. A princípio, estes novos modelos podem ser agrupados em dois grupos de diferentes abordagens: Dinâmica de fluídos baseada na interação de um número grande de partículas e dinâmica de fluídos descrita pela solução de um conjunto de equações diferenciais parciais.

Os trabalhos relacionados à primeira abordagem foram os trabalhos de Sims (1990) e Tonnesen (1991) onde os autores estudaram as forças de atração e repulsão entre as partículas para simular vários graus de viscosidade do fluído e o estado da mudança da matéria tal como o derretimento.

A segunda abordagem procura resolver diretamente um sistema de equações diferenciais parciais (PDE) descrevendo a dinâmica de fluídos (Kass e Miller, 1990). Esta técnica cria resultados realistas em termos de física, mas devido a uma simulação muito refinada

da dinâmica de fluídos requer um cálculo computacional do movimento dentro de um determinado volume de controle. Seguindo esta abordagem de pesquisa, os trabalhos desenvolvidos foram simplificando estes cálculos tornando sua utilização viável computacionalmente.

Um modelo bastante realístico pôde ser obtido usando a equação de Navier-Stokes, a mais detalhada de todos os modelos de fluídos. Devido a esta característica e algumas simplificações dos termos, a equação de Navier-Stokes tem sido amplamente utilizada na computação gráfica para simulação do movimento da água (Chen e Lobo, 1995) (Chen, Lobo, Hughes e Moshell, 1997) (Foster e Metaxas, 2000) (Witting, 1999).

O que se tem visto é que o movimento da água é um fenômeno bastante complexo e variável, incluindo efeitos difíceis de serem analisados. A descrição exata destes efeitos é normalmente caracterizada pela área da física denominada dinâmica de fluídos ou fluido-dinâmica computacional (CFD). Recentemente vêm se destacando nesta área pelo seu grande realismo e aplicações em efeitos especiais de cinema os trabalhos de Fedkiw, Stam e Jensen (2001), Geiger, Leo, Ramussen, Losasso e Fedkiw (2006), Losasso, Fedkiw e Osher (2006) e Irving, Guendelman, Losasso e Fedkiw (2006). Finalmente, seguindo a linha de Fournier e Reeves (1986) e Tessendorf (1999), pode-se citar o trabalho de Gonzato e Le Saec (2000) que também tentam simplificar a física e se preocupar com a visualização realística da cena.

3. Simulação física de oceanos

O sistema proposto para a simulação da onda na GPU é composto de 4 estágios: geração da onda, modelagem da superfície, computação óptica e renderização da água. A figura 2 mostra estes estágios e suas conexões. Os dois primeiros estágios serão realizados dentro da programação de vértice. A geração da geometria da onda será definida utilizando-se a equação de Gerstner (Tessendorf, 1999) com trajetória circular para simular a geometria em águas profundas e trajetória elíptica para simular a geometria em águas rasas. Como a proposta do trabalho é a construção de um shader na forma de efeito, a modelagem da superfície será feita de uma forma muito simples: será aplicado um shader sobre uma superfície geometricamente simples. No caso deste trabalho, será considerado um retângulo no plano XZ.

Esta é uma das vantagens da solução proposta sob o ponto de vista de reutilização e ganho em produtividade, pois pode ser facilmente incluído em qualquer game engine, apenas referenciando-se a um conjunto de shaders. Sobre o modelo, será aplicado o bump mapping para dar o efeito de rugosidade na geometria da superfície peculiar as micro ondas que se formam no oceano. Para os dois últimos estágios será desenvolvido um programa de fragmentos. A computação da óptica irá calcular a iluminação, reflexão de Fresnel e HDR. No último estágio, unindo os três anteriores e a contribuição da cor da água, finalmente será renderizado o oceano.

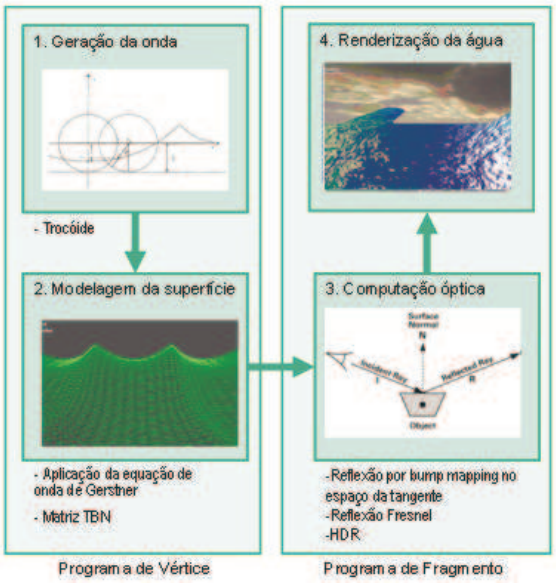


Figura 2 – Estágios do sistema de simulação

3.1 O modelo básico da onda

Usa-se aqui o modelo apresentado por Fournier e Reeves (1986). O movimento dos fluidos geralmente é descrito por duas formulações: Euleriana ou Lagrangiana. A formulação Euleriana é mais adequada na hidrodinâmica e ao estudo das ondas, especialmente para o desenvolvimento de modelos estocásticos na análise do mar. Considera-se um ponto (x, y, z) e procura-se responder questões sobre as características do fluído neste ponto em função do tempo, como por exemplo, a velocidade:

$$U = f(x, y, z, t) \qquad \text{(Eq. 2)}$$

A formulação Lagrangiana, a princípio é mais apropriada para modelagem gráfica por tratar o oceano como sendo uma primitiva geométrica. Ela descreve a trajetória de um ponto (x_0, y_0, z_0) dado por uma posição de referência. Isto pode ser visto como a trajetória de uma partícula. Por exemplo, pode-se saber a velocidade no tempo t:

$Vx=fx(x_0,y_0,z_0,t)$, $Vy=fy(x_0,y_0,z_0,t)$, $Vz=fz(x_0\ y_0,z_0,t)$ (Eq. 3)

3.2 Ondas de Gerstner

Para uma simulação efetiva do oceano, deve-se controlar a agudez (steepness) da onda. A onda perfeita tem a forma de uma senoide – como uma onda num lago calmo. Entretanto, para simular o oceano é necessário criar cristas com picos afinados e calhas

arredondadas. Para representar esta onda com mais realismo será utilizado o modelo de onda de Gerstner. O modelo físico descreve a superfície em termos de movimentos de pontos individuais na superfície.

Será considerado que uma partícula descreve um movimento circular a partir de sua posição de repouso. O plano XZ é o plano do mar em repouso e Y representa a coordenada de altura ao plano da superfície do mar. Considerando o movimento no plano XY, a equação de Gerstner simplificada será o sistema:

$$x = x_0 + r\sin(kx_0 - \omega t)$$
$$y = y_0 - r\cos(kx_0 - \omega t) \quad \text{(Eq. 4)}$$

Onde:
H = 2r é a altura da onda;
k = 2π/L é o número de onda;
L = 2π/k = gT2/2π é o comprimento de onda;
T = 2π/ω é o período;
c = L/T = ω/k é a celeridade da onda (velocidade da fase) ou seja, a velocidade de viagem da crista.

Observando-se a equação 4 como uma equação paramétrica percebe-se que se trata de uma trocóide, uma generalização da ciclóide. Esta equação representa a curva descrita por um ponto *P* que tem distância *r* do centro de um círculo de raio 1/*k* que se move circularmente sobre uma linha de distância 1/*k* sob o eixo X (figura 3). Portanto, amplitude A= r, x_0 e y_0 são as coordenadas iniciais do mar em repouso, ω é a frequência, t o tempo e k o número da onda.

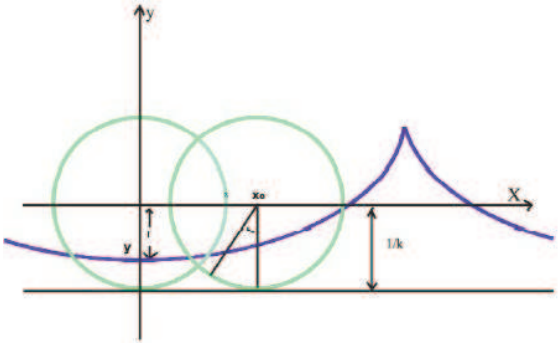


Figura 3 – Movimento de uma partícula na trocóide

Assumindo Φ = kx0- ωt como sendo a fase da onda, pode-se reescrever a equação como:

$$x = x_0 + r\sin(\Phi)$$
$$y = y_0 - r\cos(\Phi) \quad \text{(Eq. 5)}$$

Com este modelo básico é possível chegar as formas desejadas para uma cena realista. Por exemplo, alterando-se *kr*, obtem-se várias formas de onda como se pode ver na figura 4.

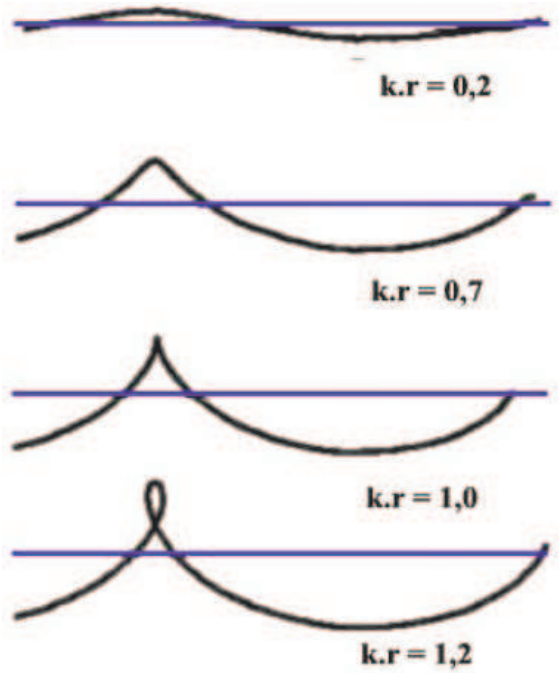


Figura 4 – Formas de onda variando o parâmetro kr.

3.3. Consideração dos estágios do sistema

A partir deste modelo básico, será definida a forma da onda bem como seu sombreamento. Desta forma, foi criado um um programa de vértice e um programa de fragmento. O programa de vértice definido implementa a equação de onda de Gerstner com variações dos atributos para simular a influência do fundo na formação da onda como a refração, influência do vento e quebra. O programa de vértice também calcula a matriz para o cálculo da iluminação. O programa de fragmento utiliza as informações recebidas do programa de vértice para calcular a iluminação e definição das cores finais.

4. Implementando a equação de Gerstner na GPU

Este trabalho utiliza a programação em GPU para implementar a simulação de onda. Esta sessão descreve cada passo necessário para obter a forma de onda final desejada.

Como o movimento de cada partícula da água é definido por um traçado circular (figura 5), com o emprego do shader é possível aplicar a equação 4 em cada vértice da figura que define a superfície do mar.

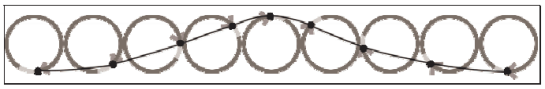


Figura 5 – Posição de uma partícula de água com o movimento da onda

Utilizando a linguagem Cg, este cálculo será efetuado no programa de vértice, que executará os seguintes passos:

1. Receber como entrada um polígono retangular representando a superfície do oceano no plano XZ;
2. Transformar os vértices do polígono utilizando a equação 4;
3. Transformar as coordenadas atuais do espaço do objeto projetado;
4. Enviar dados do vértice transformado para o processador de fragmentos.

4.1. Acrescentando a influência do vento

Uma vez definido a forma básica da onda serão acrescentados alguns efeitos especiais. Um deles é o efeito do vento sobre as ondas, fazendo com que elas sofram uma inclinação na parte superior da onda na mesma direção do vento.
Para acrescentar tal efeito, será adicionado mais um controle no ângulo de fase da equação de Gerstner:

$$\Phi = kx_0 - \omega t - \lambda \Delta y \Delta t \qquad \text{(Eq. 6)}$$

Onde:
λ é uma constante de proporcionalidade do vento
Pela expressão é possível constatar que a partícula será mais acelerada no topo e mais desacelerada na base da onda gerando uma projeção na forma da onda em direção à frente de onda.

4.2. Refração e influência do fundo do mar sobre as ondas

Quando as ondas se propagam de águas profundas para águas rasas (ao se aproximar da costa, por exemplo), as características da onda mudam assim que ela começa a sofrer a influência do solo, mantendo-se constante apenas o período. A velocidade da onda diminui com a diminuição da profundidade. Assim que as ondas passam a sentir o fundo, um fenômeno chamado refração pode ocorrer. Quando as ondas entram na zona de transição (profunda para rasa), a parte da onda em águas profundas move-se mais rapidamente que a onda em águas rasas. Como mostrado na figura 6, essa diminuição na velocidade da fase da onda pode ser percebido numa visão aérea do mar. A medida que a onda sente o fundo do mar e sua velocidade e seu comprimento de onda diminui, a crista da onda tende a se alinhar com a costa marítima. De acordo com a equação:

$$L = cT \qquad \text{(Eq. 7)}$$

Onde L é o comprimento de onda; c a celeridade da onda; T o período.

Na zona de transição entre águas profundas e águas rasas, a celeridade (c) da onda (m/s) é calculada pela equação:

$$c^2 = \frac{g}{k}\tanh(kh) \qquad \text{(Eq. 8)}$$

Onde k = 2π/L é o número de onda; g é a aceleração da gravidade e h a profundidade da água.

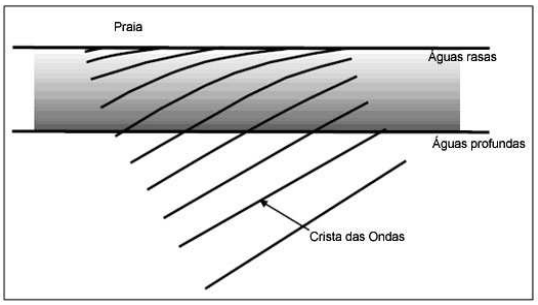


Figura 6 – Refração da onda ao se aproximar da costa (águas rasas).

Em águas profundas, onde a profundidade da água é maior que a metade do comprimento de onda, *k.h* é um valor grande de modo que tanh(*k.h*) é aproximadamente igual a 1. Portando, a celeridade em m/s pode ser escrito como:

$$c^2 = \frac{g}{k} = \frac{gL}{2\pi} \qquad \text{(Eq. 9)}$$

ou com um produto de celeridades,

$$c * c = \frac{gL}{2\pi}$$

e substituindo c = L/T, tem-se:

$$c * \frac{L}{T} = \frac{gL}{2\pi}$$

simplificando L,

$$c = \frac{gT}{2\pi}$$

ou seja, supondo g = 9,8 m/s2,
$c = T * 1{,}56$ é a celeridade da onda em águas profundas.

Em águas rasas, onde a profundidade da água (h) é menor que 1/20 do comprimento de onda, *t.h* é um valor pequeno, logo tanh(*t.h*) é aproximadamente igual a *t.h*. Substituindo tanh(*t.h*) por *t.h* na equação 8 e simplificando k, tem-se:

$$c^2 = \frac{g}{k}(kh) = gh \qquad \text{(Eq. 10)}$$

Extraindo a raiz quadrada, obtém-se:
$c = \sqrt{gh}$, como sendo a celeridade da onda em águas rasas. A figura 7 mostra a variação das ondas representada nas equações 9 e 10.

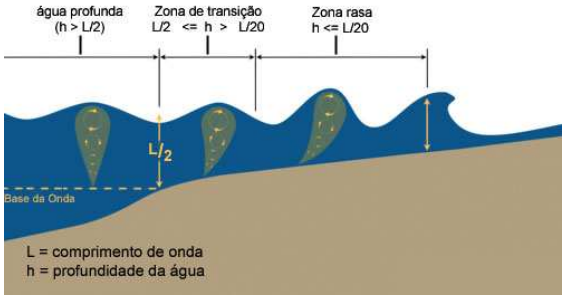


Figura 7 – Transição e variação das ondas

4.3. O comprimento de onda de acordo com a profundidade

A diminuição da profundidade da água também altera o comprimento de onda, sendo que o período permanece constante (Kinsman, 1965). Se chamarmos de k_∞ o número de onda em uma profundidade infinita, uma boa aproximação para o número de onda k na profundidade h é dada por:

$$k\tanh(kh) = k_\infty \qquad \text{(Eq. 11)}$$

Quando x → 0, então tanh(x) → x. Logo, em águas rasas, onde a profundidade é pequena, a relação fica:

$$k^2 h = k_\infty \quad \text{ou} \quad k = \sqrt{\frac{k_\infty}{h}}$$

Quando x → ∞, então tanh(x) → 1, logo k → k∞. Uma vez que $k.h = 2\pi h/L$, com uma relação h/L de 1/2 obtém-se um argumento de π para a tangente hiperbólica o que é praticamente igual a 1. Por esta razão, a "profundidade" tem relacionamento com o comprimento de onda e significa a relação h/L ser maior que 1/2 como citado anteriormente na figura 7. Assim, uma boa aproximação para a equação 11 é:

$$k = \frac{k_\infty}{\sqrt{\tanh(k_\infty h)}} \qquad \text{(Eq. 12)}$$

4.4. A celeridade em relação à profundidade

Uma vez afetado o comprimento de onda, a celeridade da onda também é afetada como mostrado no item anterior, o que significa: $c/c_\infty = k_\infty/k$ e a onda é refratada assim como a velocidade sofre diminuição. De fato, pode-se aplicar a lei de Snell Descartes (USP-Educar, 2006) para calcular o ângulo que a frente de onda faz ao partir de uma profundidade infinita e entrar numa profundidade h.

$$\frac{\sin(\theta_h)}{\sin(\theta_\infty)} = \frac{c_h}{c_\infty} \qquad \text{(Eq. 13)}$$

A figura 8 mostra que quando a onda emitida por A' se desloca até B em um intervalo de tempo t, a onda emitida por A, neste mesmo intervalo de tempo, sofre um deslocamento menor até B', considerando que v2 < v1.

Sendo: A'B = v1.t e AB' = v2.t
Obtem-se:

$$\frac{A'B}{AB'} = \frac{v_1}{v_2}$$

Da geometria da figura 7, tem-se:

$$\sin\theta_1 = \frac{A'B}{AB} \qquad \sin\theta_2 = \frac{AB'}{AB} \qquad \text{(Eq. 14)}$$

Dividindo as duas equações, obtém-se:

$$\frac{\sin\theta_1}{\sin\theta_2} = \frac{A'B}{AB'} = \frac{v_1}{v_2}$$

Como n1 = vc / v1 e n2 = vc / v2, substituindo na equação anterior, obtem-se a expressão da lei de Snell Descartes:

$$\frac{\sin(\theta_1)}{\sin(\theta_2)} = \frac{n_2}{n_1}$$

O efeito de profundidade não pode ser calculado considerando-se apenas informações locais, o atraso da fase que é introduzido é acumulativo. Sendo k é uma função de profundidade, que por sua vez é função de x0. Assumindo Φ = 0 para x0 = 0, e que a constante de proporcionalidade λ = 0, a equação da fase agora será:

$$\Phi = -\omega t + \int_0^{x_0} k(x)dx \qquad \text{(Eq. 15)}$$

onde

$$k(x) = \frac{k_\infty}{\sqrt{\tanh(k_\infty h(x))}}$$

Desta forma será possível simular o efeito de refração da onda ao aproximar-se da costa e receber influência do fundo do mar.

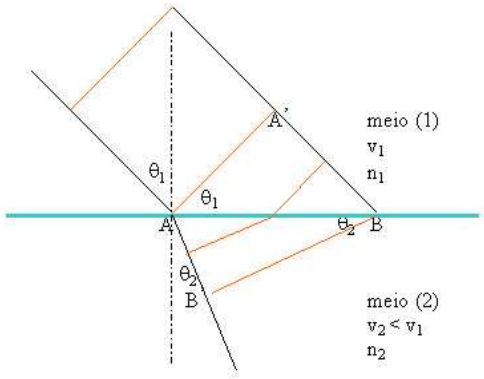


Figura 8 – Frente de onda na refração de acordo com a lei de Snell.

4.5. Quebra da onda na costa

As teorias clássicas dizem que a medida que a onda se aproxima da costa, sua trajetória passa a ser elíptica ao invés da circular em águas profundas. Biesel (1952) propôs um modelo em que o eixo maior da elipse se

alinhe com a inclinação do fundo do mar até que a profundidade se torne igual a zero (figura 9 e 10).

Figura 9 – Como a profundidade afeta a órbita

Figura 10 – Como a profundidade afeta a forma da onda

Para adaptar a programação em GPU será utilizada a forma simplificada de Fournier-Reeves (1986) levando em consideração o custo computacional:

$$x = x_0 + r\cos\alpha.S_x.\sin\Phi + \sin\alpha.S_z.\cos\Phi$$
$$y = y_0 - r\cos\alpha.S_z.\cos\Phi + \sin\alpha.S_x.\sin\Phi \quad \text{(Eq. 16)}$$

Onde Φ é a fase; senα = sen γe-k_0h , γ é a inclinação do fundo do mar em direção a trajetória da onda; Sx = 1/ (1 – e-kxh) é o incremento do eixo maior da elipse; Sy = Sx(1 - e-kyh) é o decremento do eixo menor da elipse; K_0 determina a influência da profundidade na inclinação na elipse; Kx é um fator de enlargamento do eixo maior da elipse; Ky é um fator de redução do eixo menor da elipse; r é o raio do disco.

É importante notar que Sx→∞ quando a profundidade h → 0.

Com essa mudança da equação do movimento é possível obter um melhor realismo na forma da onda quando ela se quebra aproximando-se da costa e recebe a influência do fundo do mar.

4.6. Correção do modelo de Fournier-Reeves

O modelo apresentado anteriormente apresenta duas limitações. A primeira delas é que o modelo não permite que o fundo do mar tenha inclinações negativas, pois as ondas quebram-se na direção reversa da propagação da onda. Como pode ser visto na figura 11, esta limitação torna-se muito irreal. Já que uma possibilidade de fundo irregular com depressões e saliências é muito comum na natureza, especialmente em regiões rochosas ou em águas muito agitadas.

Figura 11 – Limitação do modelo de Fournier-Reeves.

Para resolver este problema usando o modelo de Fournier-Reeves, é necessário limitar a inclinação do eixo maior da elipse para um ângulo sempre positivo, considerado zero as inclinações negativas.

O segundo problema vem do fato que no modelo de Gerstner, os círculos descritos pelas partículas de água são restritos ao raio do disco, isto é, r <= 1/k. Quando r > 1/k podem ocorrer laços na equação que representa a forma da onda que não ocorre na natureza como mostrado na figura 12. Neste caso, a solução consiste em fazer com que o eixo maior da elipse receba um valor maior que o disco de raio r. Esta ação simula o aparecimento da quebra da onda em profundidade média, mas ao aproximar-se da costa, o modelo torna-se inoperante, pois são formados laços na forma da onda. Para este problema, a solução utilizada neste trabalho foi limitar o eixo maior da elipse ao raio r do disco adquirindo a segunda forma mostrada na figura 12.

Figura 12 – Correção de laços na formação da onda.

Para melhorar a aparência da geometria da queda da onda, aproximando-se das ondas em espiral (quebras mergulhando), Gonzato e Le Saec (1997) propuseram uma alteração na fase da equação de Fournier-Reeves esticando e torcendo a crista da onda como mostra a figura13. Gonzato adicionou três novas funções chamadas Strech, Orientation e Displacement. A função Strech é usada para simular a aceleração da partícula na crista da onda. As funções Orientation e Displacement são combinadas para simular a influência da força da gravidade.

Figura 13 – Perfil da onda de Gonzato e Le Saec (1997)

4.7. A função Strech

Para manter a forma inicial da onda de Biesel, foi adicionado como parâmetro da função, um fator para esticar a crista da onda em direção ao eixo maior da elipse ao passo que nenhuma modificação é realizada na calha da onda. O parâmetro St_{max} define este tamanho máximo de enlargamento. Também é criado um parâmetro de escala K_s para determinar a influência da profundidade na função Strech .

Uma função parabólica chamada $Strech(\Phi, \lambda)$ de fase Φ é usada. O fundo da onda é obtido através do valor mínimo de y na equação paramétrica da elipse da seguinte forma:

$$\Phi_{\min} = a\tan\left(\frac{\sin(\alpha)S_x}{\cos(\alpha)S_y}\right)$$

E a função Strech é definida como:

$$Strech(\phi, St_{\max}) = \frac{1}{\pi^2}\left(St_{\max}\phi^2 - 2St_{\max}\Phi_{\min}\phi + St_{\max}\Phi_{\min}^2\right)$$

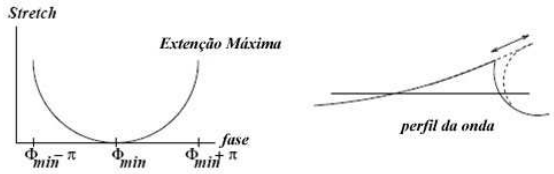


Figura 14 – Função Strech

4.8. Quebrando a onda

Em complemento a função Strech, é necessário alterar a forma da onda para representar a força da gravidade fazendo com que a mesma se quebre. Para isso, é criado uma função de orientação da crista da onda. Esta função altera a forma da onda e progressivamente adiciona um novo estiramento que decrementa na direção do eixo maior da elipse até a posição vertical.

Esta função de orientação é combinada com uma função de deslocamento progressivo. Este valor é limitado a 2*r* de modo que a não permitir a colisão da crista com o fundo da onda. Além disso, um fator de escala chamado K_d é utilizado para determinar a influência da profundidade na função de deslocamento. No topo da crista da onda, a velocidade é importante, mas logo abaixo dela a velocidade é bem menor. Deste modo, esta função é dividida em três partes de forma empírica:

-

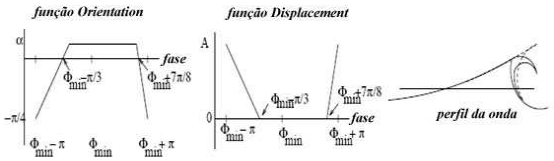


Figura 15 – Quebra da onda

Acrescentando a função *Strech*, a função *Displacement* e a função *Orientation*, a equação da onda se transforma em:

$$\begin{cases} x = x_0 + R\tau'_\beta S_x \sin(\Phi) + R\tau_\beta S_y \cos(\Phi) + Stretch(\Phi, St_{\max})\tau'_\beta e^{-K_s h} \\ + Displacement(\Phi, 2r)\cos(Orientation(\Phi, \beta))e^{-K_d h} \\ y = y_0 - R\tau'_\beta S_y \cos(\Phi) + R\tau_\beta S_y \sin(\Phi) + Stretch(\Phi, St_{\max})\tau'_\beta e^{-K_s h} \\ + Displacement(\Phi, 2r)\sin(Orientation(\Phi, \beta))e^{-K_d h} \end{cases}$$

Onde:

$\tau_\beta = \sin(\beta)e^{-0.1h}, \tau'_\beta = \sqrt{1-\tau_\beta}$, sendo β é a inclinação do fundo;

$S_x = \frac{1}{1-e^{-0.11h}}$, é o incremento do eixo maior;

$S_y = S_x(1-e^{-0.09h})$, é o decremento do eixo menor;

5. Visualização

O capítulo anterior definiu a modelagem geométrica da onda.. Como a água pode apresentar-se com uma aparência diferente dependendo do contexto da cena, é importante definir algumas categorias de efeitos de água. No caso deste trabalho, foram utilizadas técnicas para que a água assemelhe-se a aparência do oceano, mas não houve contribuição inédita neste tema, sendo que as técnicas aplicadas foram reflexão por environment mapping, efeito Fresnel e HDR.

6. Implementação e resultados

O objetivo deste trabalho consiste em criar uma simulação realística da superfície das ondas da superfície do mar. Para criar uma ferramenta produtiva e reutilizável, foi criado um efeito (arquivo.fx) de modo a simplificar a sua implementação tanto para um programador quanto para um artista 3D.
Utilizou-se o FX Composer 1.8 que é uma IDE para desenvolvimento de shader da NVIDIA® na linguagem Cg (Shader 2.0), e também foi testado aplicando-se o efeito dentro de um software de renderização 3D proprietário, o 3D MAX® da Discreet®. Todas as fotos apresentadas na figura 16 são retiradas da aplicação sendo executada em tempo-real.

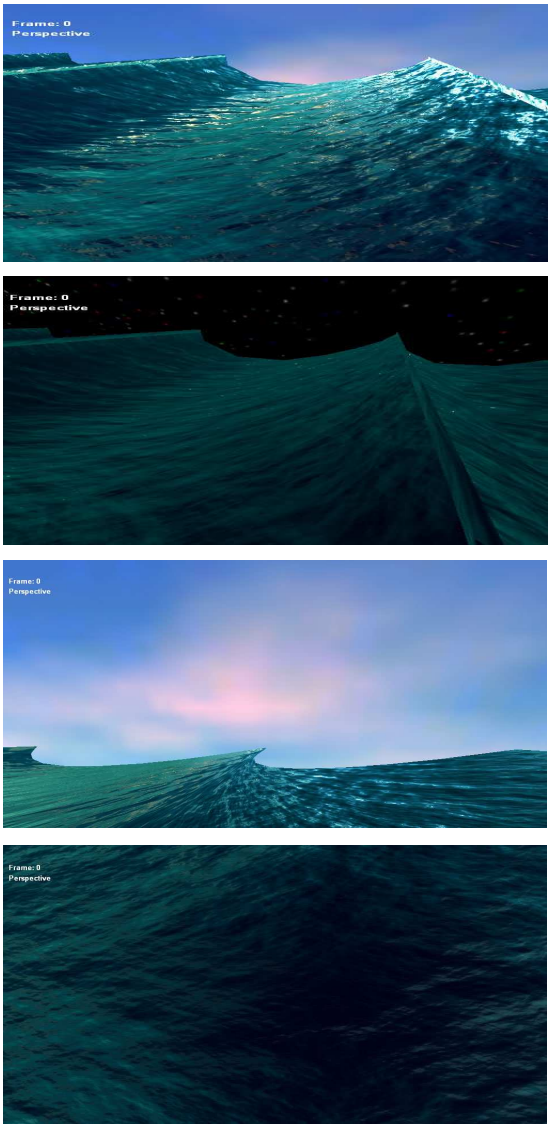


Figura 16 – Resultados em tempo-real

Para visualizar a cena, foi utilizado um computador AMD Athlon XP 1800 com 512 RAM com uma placa de vídeo NVIDIA FX 5200 Shader 2.0.

O shader foi programado de forma que alguns parâmetros pudessem ser configurados pelo usuário. Estes parâmetros definem o comportamento visual e físico da onda do mar. De forma empírica e visual foram configurados valores para estes parâmetros de forma a atingir o objetivo final de uma cena realista. Manipulando-se essas propriedades, é possível alterar a granularidade das ondas capilares; alterar a velocidade da partícula da água, alterar a contribuição da reflexão de Fresnel e HDR no cálculo da iluminação; criar ambientes noturnos, diurnos, sol, estrelas através do cubemap; alterar o período, a freqüência e raio da onda e controlar a contribuição do vento.

7. Conclusão

Foi proposto um sistema de visualização tempo real para simulação de ondas oceânicas. Para alcançar um nível de simulação que caracterizasse uma aplicação em tempo real, todo o processamento e programação deste trabalho foi direcionado para a arquitetura e o hardware gráfico da GPU.

Foi abordado o comportamento das ondas em águas profundas, águas rasas com a influência do fundo do mar sobre a forma destas ondas. Foi possível simular a modelagem geométrica baseada nas leis físicas e algumas configurações empíricas com atenção para as ondas se quebrando na costa oceânica. Simulou-se também a refração das ondas quando esta sente o fundo do mar e perde velocidade de fase de acordo com a lei de Snell Descartes, alterando as propriedades da onda tais como o comprimento de onda e velocidade, a medida que se aproxima da costa.

Os resultados obtidos foram satisfatórios e convincentes quanto a visualização da superfície de um oceano com formação de ondas. Para aumentar o desempenho da renderização, pode-se acrescentar algoritmos de níveis de detalhe (LOD) para que os polígonos distantes da visão possam ser renderizados com menos detalhes (Bustamante e Celes, 2002).

Para representar a topologia do fundo do mar que foi utilizado nas equações da onda, podería-se utilizar um mapa de altura que armazenasse estas informações. Assim o programa de vértice poderia ler este mapa e interpretar os valores tornando mais refinado a visualização. Entretanto, este recurso só é possível com o perfil do Shader 3.0 (Gerasimov, Fernando e Green, 2004) utilizando o recurso de Vertex Texture. Neste trabalho não foi gerado o efeito de espuma e bolhas na superfície das ondas. O que pode ser gerado utilizando a física de sistema de partículas e autômatos celulares. Pode-se também gerar o fenômeno de caustics que ocorre na superfície da água (Fernando, 2004).

A crista da onda ficou com uma aparência relativamente regular e reta que não existe na natureza da onda. Seria interessante criar irregularidades e ruídos na formação dessas cristas gerando padrões aleatórios utilizando a função noise.

8. Bibliografia

(Adabala e Manobar, 2002) N. Adabala, S. Manohar, Techniques for realistic visualization of fluids: a survey, Comput. Graph. Forum volume 21 (1), pp. 65-81, 2002.

(Biesel, 1952) F. Biesel, Study of wave propagation in water of gradually varying Depth, Gravity Waves, pp. 243-253, U.S. National Bureau of Standards Circular 521, 1952.

(Blinn, 1978) J.F. Blinn, Simulation of wrinkled surfaces, Proceedings of SIGGRAPH'78, Comput. Graph, volume 12 (3), pp. 286–292. 1978.

(Chen e Lobo, 1995) J. Chen, N. Lobo, Toward interactive-rate simulation of fluids with moving obstacles using navier-stokes equations, Graphical models and Image Processing, pp.107-116, (março 1995).

(Chen, Lobo, Hughes e Moshell, 1997) J.X. Chen, N.V. Lobo, C.E. Hughes, J.M. Moshell, Real-time fluid simulation in a dynamic virtual environment, IEEE Comput.Graph. Appl., pp.52–61, (maio-junho 1997).

(Fedkiw, Stam e Jensen, 2001) R .Fedkiw, J. Stam., H. W. Jensen. Visual Simulation of Smoke, Proceedings of SIGGRAPH 2001, pp. 15-22, 2001.

(Ferwwerda, 1999) J.A. Ferwerda, Three varieties of realism in computer graphics, Cornell Workshop on Rendering, Perception and Measurement, 1999.

(Fernando, 2004) R. Fernando. GPU Gems: Programming Techniques, Tips, and Tricks for Real-Time Graphics, Addison-Wesley Pub, 2004.

(Fishman e Schachter, 1980) B. Fishman, B. Schachter, Computer display of height fields, Comput. Graph. 5 pp. 53–60, 1980.

(Fournier e Reeves, 1986) A. Fournier, W. T. Reeves, A simple model of ocean waves, SIGGRAPH´86, volume 20, pp. 75-84, 1986.

(Foster e Metaxas, 2000) N. Foster, D. Metaxas, Modeling water for computer animation, Commun. ACM volume 43 (7), pp. 60–67, 2000.

(Foster e Fedkiw, 2001) N. Foster, R. Fedkiw, Practical Animation of Liquids, Proceedings of SIGGRAPH 2001, pp. 23-30, 2001.

(Geiger, Leo, Ramussen, Losasso e Fedkiw, 2006) Geiger, W., Leo, M., Rasmussen, N., Losasso, F. and Fedkiw, R., So Real It'll Make You Wet, SIGGRAPH 2006 Sketches and Applications, 2006.

(Gerasimov, Fernando e Green, 2004) P. Gerasimov, R. Fernando, S. Green, Shader Model 3.0: Using Vertex Texture, NVidia Corporation, 2004.

(Gerstner, 1809) F.J. Gerstner, Theorie der wellen, Ann. der Physik 32, pp. 412-440, 1809.

(Gonzato, Le Saec, 2000) J.C. Gonzato, B. Le Saec, On Modeling and Rendering Ocean Scenes, Journal of Visualisation and Computer Simulation, 11, pp. 27-37, 2000.

(Irving, Guendelman, Losasso e Fedkiw, 2006) Irving, G., Guendelman, E., Losasso, F. and Fedkiw, R., Efficient Simulation of Large Bodies of Water by Coupling Two and Three Dimensional Techniques, SIGGRAPH 2006, ACM TOG 25, pp. 805-811, 2006.

(Imamiya e Zhang, 1995) A. Imamiya, D. Zhang, Modelling breaking ocean waves, influence of floor and refraction, Pacific Graphics 95, 1995.

(Iglesias, 2004) A. Iglesias, Computer graphics for water modeling and rendering: a survey, Future Generation Computer System, volume 20 (8), pp. 1355-1374, (novembro 2004).

(Kass e Miller, 1990) M. Kass, G. Miller, Rapid, stable fluid dynamics for computer graphics, ACM Press, New York, NY, USA, 1990.

(Kinsman, 1965) B. Kinsman, Wind Waves: Their Generation and Propagation on the Ocean Surface, Prentice Hall, 1965.

(Peachey, 1986) D. R. Peachey, Modeling waves and surf, SIGGRAPH´86, volume 20, pp. 65-70, 1986.

(Reeves, 1983) W.T. Reeves, Particle systems—a technique for modeling a class of fuzzy objects,: Proceedings of SIGGRAPH'83, Comput. Graph. Volume 17 (3) pp. 359–376, 1983.

(Sims, 1990) K. Sims, Particle animation and rendering using data parallel computation, Proceedings of SIGGRAPH'90, Comput.Graph. volume 24 (4), pp. 405–413, 1990.

(Ts´o e Barsky, 1987) P. Y. Ts'o, B. A. Barsky, Modeling and rendering waves: Wave-tracing using beta-splines and reflective and refractive texture mapping,. ACM Transactions on Graphics, volume 6 pp. 191-214, (Julho 1987).

(Tonnesen, 1991) D. Tonnesen, Modeling liquids and solids using thermal particles,: Proceedings of Graphics Interface'91, pp. 255–262, 1991.

(Tessendorf, 1999) J. Tessendorf, Simulating ocean water, SIGGRAPH'99 Course Notes, 1999.

(Witting, 1999) P. Witting, Computational fluid dynamics in a traditional animation environment, in: Proceedings of SIGGRAPH'99, pp. 129–136, 1999.

(Wloka, 2002) M. Wloka, Technical report: Fresnel Reflexion, NVIDIA Corporation, (Junho 2002).

The GPU Used as a Math Co-Processor in Real Time Applications

Marcelo P. M. Zamith
Esteban W. G. Clua
Aura Conci
Anselmo Montenegro
Instituto de Computação
Universidade Federal Fluminense

Paulo A. Pagliosa
Departamento de Computação e Estatística
Universidade Federal de Mato Grosso do Sul

Luis Valente
VisionLab/IGames
Departamento de Informática
PUC-Rio

Abstract

This paper presents the use of GPU as a math co-processor in real-time applications, in special games and physics simulation. Game loop architecture is used to validate the use of GPU such as math and physics co-processor, thus it is shown by this paper a new game loop architecture that employs graphics processors (GPUs) for general-purpose computation (GPGPU). A critical issue in this field is the process distribution between CPU and GPU. The presented architecture consists in a model for distribution and our implementation showed many advantages in comparison to other approaches without a GPGPU stage.

The architecture presented here was mainly designed to support mathematics and physics on the GPU, therefore the GPU stage in the model proposed works to help the CPU such as a math co-processor. Nevertheless, any kind of generic computation can be adapted. The model is implemented in an open source game engine and results obtained using this platform are presented.

Keywords:: game loop, GPGPU, co-processor.

Author's Contact:

{mzamith,esteban,aconci,anselmo}@ic.uff.br
*pagliosa@dct.ufms.br
*lvalente@inf.puc-rio.br

1 Introduction

New graphics hardware architectures are being developed with technologies that allow more generic computation. GPGPU (general-purpose computation on GPUs) became very important and started a new area related to computer graphics research. This leads to a new paradigm, where the CPU — central processing unit — does not need to compute every non-graphics application issue. However, not every kind of algorithm can be allocated for the GPU — graphics processing unit — but only those that can be reduced to a stream based process. Besides, even if a problem is adequate for GPU processing, there can be cases where using the GPU to solve such problems is not worthy, because the latency generated by memory manipulation on the GPU can be too high, severely degrading application performance.

Many mathematics and physics simulation problems can be formulated as stream based processes, making it possible to distribute them naturally between the CPU and the GPU. This may be extremely useful when real time processing is required or when performance is critical. However, this approach is not always the most appropriate for a process that can be potentially solved using graphics hardware. There are many factors that must be considered before deciding if the process must allocate the CPU or the GPU. Some of these factors may be fixed and some may depend on the process status.

A correct process distribution management is important for two reasons:

- It is desired that both the GPU and the CPU have similar process load, avoiding the cases where one is overloaded and the other is fully idle;
- It is convenient to distribute processes considering which architecture will be more efficient for that kind of problem.

For instance, many real time graphics applications render the scene more than 60 frames per second, but execute physics or artificial intelligence simulations less frequently. Almost every video display has a refresh rate of 60 Hz, when rendering more than 60 frames per second, many of the calculated frames are discarded. In this case, it would be convenient to reduce graphics calculations and increase the other ones.

As it is not always possible to tell which processing route (GPU or CPU) a problem should go through, it is important that the framework or engine being used for the application development to be responsible for dynamically allocating jobs.

This paper proposes a new architecture for a correct and efficient semi-automatic process load distribution, considering real time applications like games or virtual reality environments. The use of GPU in this model has focus to help the CPU at processing of mathematics and physics such as mathematic co-processor. The presented idea is implemented in an academic open source game engine [Valente 2005].

The paper is organized as follows. Section 2 summarizes the main functionalities of a physics engine currently being developed and that is integrated into the academic framework, as a component. The purpose of that section is to identify which math operations are more suitable for implementation on GPU. Section 3 presents related works. Section 4 describes the architecture of framework adopted and its GPGPU shader support, as well as a first model for distributing tasks between CPU and GPU. Section 5 presents a simple example and results. Finally, Section 6 points out conclusions and future works.

2 Math and Physics on GPU

The ODE (Open Dynamics Engine) was integrated to the framework by the authors. This open source component is responsabile for real time dynamic simulation of rigid and elastic bodies and corresponds to one of the state-of-the-art tools available for physics simulation. A rigid body is a (possibly continuum) particle system in which the relative distance between any two particles never changes, despite the external forces acting on the system. The *state* $\mathbf{X}(t)$ of a body at time t is defined as [Feijó et al. 2006]:

$$\mathbf{X}(t) = \begin{bmatrix} \mathbf{x}(t) \\ \mathbf{q}(t) \\ \mathbf{P}(t) \\ \mathbf{L}(t) \end{bmatrix}, \tag{1}$$

where $\mathbf{X}$ is the world position of the center of mass of the body, $\mathbf{q}$ is a quaternion that represents the rotation of the local reference frame of the body in relation to the world frame, $\mathbf{P}$ is the world linear momentum and $\mathbf{L}$ is the world angular momentum of the body. The main functionality of a physics engine is, known the state $\mathbf{X}(t)$, to determine the next state $\mathbf{X}(t + \Delta t)$ of each rigid body into the scene, where Δt is the time step. This task involves the integration of the *equation of motion* of a rigid body [Feijó et al. 2006]:

$$\frac{d}{dt}\mathbf{X}(t) = \begin{bmatrix} \mathbf{v}(t) \\ \frac{1}{2}\mathbf{w}(t)\mathbf{q}(t) \\ \mathbf{F}(t) \\ \boldsymbol{\tau}(t) \end{bmatrix}, \tag{2}$$

where $\mathbf{v} = m^{-1}\mathbf{P}$ is the world linear velocity of the center of mass of the body, $\mathbf{w}$ is the quaternion $[0, \boldsymbol{\omega}]$, $\boldsymbol{\omega} = \mathbf{I}^{-1}\mathbf{L}$ is the angular velocity of the body, $\mathbf{F}$ is the external force and $\boldsymbol{\tau}$ is the external

torque applied to the body, and m and $\mathbf{I}$ are the mass and *inertia tensor* of the body at time t, respectively.

Equation (2) is a first order ordinary differential equation (ODE); the component of a physics engine responsible by its integration (usually by applying a numerical method such as Runge-Kutta fourth-order) is called *ODE solver*.

Generally, the motion of a rigid body is not free, but subject to *constraints* that restraint one or more *degrees of freedom* (DOFs) of the body. Each constraint applied to a body introduces an unknown *constraint force* that should be determined by the physics engine in order to assure the restriction of the corresponding DOF. Constraints can be due to *joints* between (usually) two bodies and/or (*collision* or *resting*) *contact* between two or more bodies [Feijó et al. 2006].

In order to compute the contact forces that will prevent interpenetration of bodies, a physics engine needs to know at time t the set of *contact points* between each pair of bodies into the scene. The contact information includes the position and surface normal at the contact point, among others. This task is performed by a component integrated to the engine responsible for *collision detection*, which can be divided in a *broad* and a *narrow* phase [Ericson 2005]. In the broad phase only a sort of (hierarchies of) bounding volumes containing the more complex geometric shapes of the bodies are checked for intersecting; if they do not intersect, their bodies do not collide. Otherwise, in the narrow phase the shapes themselves are checked for intersecting and the contact information is computed.

Once found the contact points, the physics engine must simultaneously compute both the contact and joint forces and applies them to the respective bodies. Mathematically, this can be formulated as a *mixed linear complementary problem* (LCP), which can be solved, for example, by using the Lenke's algorithm [Eberly 2004]. The task is performed by a component of the engine called *LCP solver*.

In short, the main tasks performed at each time step during the simulation of a scene made of rigid bodies are: collision detection, collision handling, and resolution of differential equations.

For collision detection, GPUs can be used as a co-processor for accelerating mathematics or for fast image-space-based intersection techniques [Ericson 2005]. These ones rely on rasterizing the objects of a collision query into color, depth, or stencil buffers and from that performing either 2D or 2.5D overlap tests to determine whether the objects are in intersection [Baciu and Wong 2003; Govindaraju et al. 2003; Heidelberger et al. 2004]. Section 5 presents an illustrative implementation.

The ODE solver is a component that can be also efficiently implemented on GPU, since the integration of Equation (2) for a rigid body is performed independently of the other ones (therefore in parallel). Besides, data in Equations (1) and (2), which are related to the state of a rigid body and its derivative, can be easily write to and read from streams; the implementation of a method such as a Runge-Kutta solver on GPU (the arithmetic kernel) is also straightforward.

In the literature there are many works on solving ODEs on GPU, especially those related to (discrete) particle system simulation. For example, Kipfer et al. presented a method for this including inter-particle collisions by using the GPU to quickly sort the particles to determine potential colliding pairs [Kipfer et al. 2004]. In a simultaneous work, Kolb et al. produced a GPU particle system simulator that supported accurate collisions of particles with scene geometry by using GPU depth comparisons to detect penetration [Kolb et al. 2004]. Related to particle systems is cloth simulation, Green demonstrated a very simple cloth simulation using Verlet integration with basic orthogonal grid constraints [Green 2003]. Zeller extended this work with shear constraints that can be interactively broken by the user to simulate cutting of the cloth into multiple pieces [Zeller 2005].

Realistic physical simulation of deformable solid bodies is more complicated than rigid ones and involves the employment of numerical methods for solving the partial differential equations (PDEs) that govern the behavior of the bodies. Such methods are based on a subdivision of the volume or surface of a solid in a mesh of discrete elements (e.g. tetrahedrons or triangles); mathematically, the PDEs are transformed in systems of equations that, once solved, give the solution of the problem at vertices of the mesh.

Two domain techniques are the finite differences and finite element methods (FEM). The former has been much more common in GPU applications due to the natural mapping of regular grids to the texture sampling hardware of GPUs. Most of this work has focused on solving the pressure-Poisson equation that arises in the discrete form of the Navier-Stokes equations for incompressible fluid flow. The earliest work on using GPUs to solve PDEs was done by Rumpf and Strzodka [Rumpf and Strzodka 2005], where they discuss the use of finite element schemes for PDE solvers on GPUs in detail.

The current research of some of the authors on deformable bodies initially considers perfect linear solids only, which are governed by Navier equation. The solving technique is based on the boundary element method (BEM) with use of the Sherman-Morrison-Woodbury formula to achieve real time responses, as suggested in the work of James and Pai [James and Pai 1999]. One of the possibilities of GPGPU is to use the GPU as a math co-processor for implementation of a number of techniques for numerical computing, mainly those ones for matrix operations and solving linear systems of equations.

The design of an architecture for process distribution is a critical issue for an efficient collaboration between CPU and GPU. The present work implements some of the mentioned calculations in order to validate and test this distribution strategy. Very good results are achieved as discussed in Section 5.

3 Related Works

GPGPU is a research area in expansion and much promising early work has appeared in the literature. Owens et al. present a survey on GPGPU applications, which range from numeric computing operations, to non-traditional computer graphics processes, to physical simulations and game physics, and to data mining, among others [Owens et al. 2007]. This section cites works related to math and physics of solids on GPU.

Bolz et al. [Bolz et al. 2003] presented a representation for matrices and vectors on GPU. They implemented a sparse matrix conjugate gradient solver and a regular grid multigrid solver for GPUs, and demonstrated the effectiveness of their approach by using these solvers for mesh smoothing and solving the incompressible Navier-Stokes equations.

Krüger and Westermann took a broader approach and presented a general linear algebra framework supporting basic operations on GPU-optimized representations of vectors, dense matrices, and multiple types of sparse matrices [Krüger and Westermann 2003]. Using this set of operations, encapsulated into C++ classes, Krüger and Westermann enabled more complex algorithms to be built without knowledge of the underlying GPU implementation.

Galoppo et al. [Galoppo et al. 2005] presented an approach to efficiently solve dense linear systems. In contrast to the sparse matrix approaches, they stored the entire matrix as a single 2D texture, allowing them to efficiently modify matrix entries. The results show that even for dense matrices the GPU can to outperform highly optimized ATLAS implementations.

Fatahalian et al. did a general evaluation of the suitability of GPUs for linear algebra operations [Fatahalian et al. 2004]. They focused on matrix-matrix multiplication and discovered that these operations are strongly limited by memory bandwidth when implemented on the GPU. They explained the reasons for this behavior and proposed architectural changes to improve GPU linear algebra performance.

Full floating point support in GPUs has enabled the next step in physically based simulation: finite difference and finite element techniques for the solution of systems of partial differential equations. Spring-mass dynamics on a mesh were used to implement basic cloth simulation on a GPU [Green 2003; Zeller 2005].

Recently, NVIDIA and Havok have been shown that rigid body simulations for computer games perform very well on GPUs [Bond 2006]. They demonstrated an API, called Havok FX, for rigid body and particle simulation on GPUs, featuring full collisions between rigid bodies and particles, as well as support for simulating and rendering on separate GPUs in a multi-GPU system. Running on a PC with dual NVIDIA GeForce 7900 GTX GPUs and a dual-core AMD Athlon 64 X2 CPU, Havok FX achieves more than a 10 times speedup running on GPUs compared to an equivalent, highly optimized multithreaded CPU implementation running on the dual-core CPU alone.

Havok FX supports a new type of rigid body object called debris primitive. This one is a compact representation of a 3D object on possible collision that can be processed via shader model 3.0 (SM3.0) in a very efficient manner. Debris primitives can be premodeled as part of a game's static art content or generated on the fly during game play by the CPU, based on the direction and intensity of a force (e.g. brick and stone structure blown apart by a cannon blast). Once generated by the CPU, debris primitives can be dispatched fully to the GPU for physical simulation and final rendering. Debris primitives can also interact with game-play critical objects, through an approach that provides the GPU with a one-way transfer of critical information that allows debris primitives to respond to game-play objects and large-scale world definitions.

It is important to mention that, despite the number of works devoted to GPGPU available in literature, no work deals with the issue of distribution of tasks between CPU and GPU, as proposed here.

4 GPU-CPU Process Distribution Model

For games and other real-time visualization and simulation applications, there are many different known loop models, such as the simple coupled, synchronized coupled and the multithread uncoupled [Valente et al. 2005].

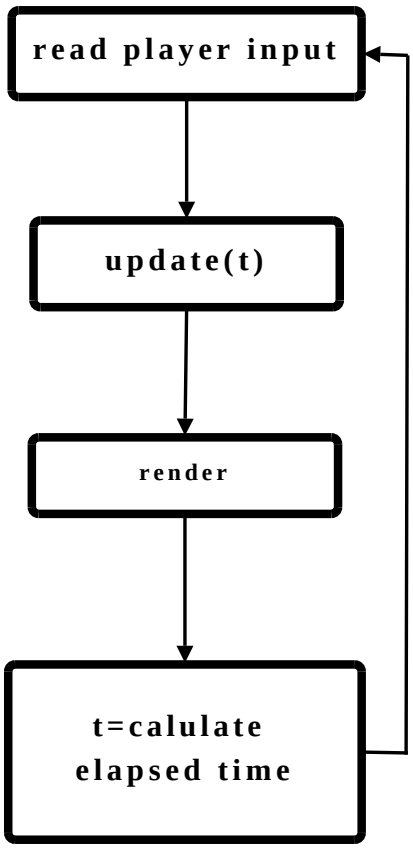


Figure 1: Simple coupled model.

Basically, these architectures arrange typical processes involved in a game in different manners, but always inside a main loop. The processes consist of the following ones: data input (from keyboard, joystick, mouse, etc.), update — especially physics, artificial intelligence (AI), and application logic — and rendering.

In the simple coupled model, the stages are arranged sequentially, as shown in Figure 1.

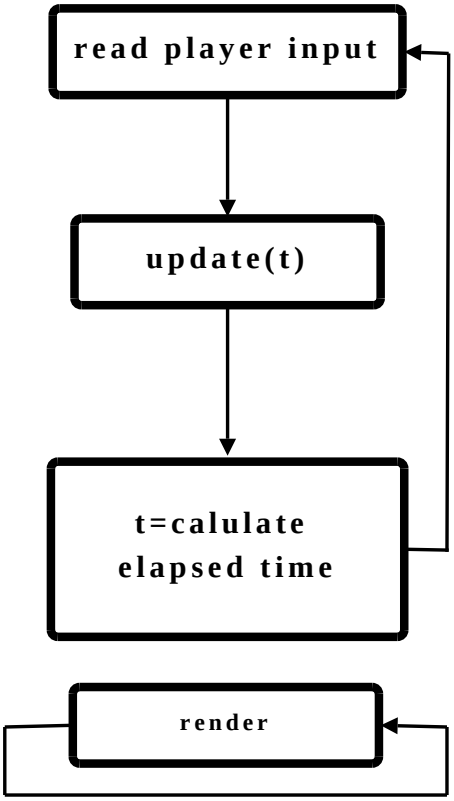


Figure 2: Multithread uncoupled model.

The multithread uncoupled model, depicted in Figure 2, separates the update and rendering stages in two loops, so they can run independently from each other. In this model, the input and update stages can run in a thread, and the rendering stage can run in another thread.

As presented before, the described models comprise three stages: input, update, and rendering. With the possibility of using the GPU for generic computation, this paper proposes a new model, called *multithread uncoupled with GPGPU*. This model is based on the multithread uncoupled model with the inclusion of a new stage, defined as GPGPU. This architecture is composed of threads, one for the input and update stages, another for the GPGPU stage, and the last one for the rendering stage. Figure 3 depicts a schematic representation for this approach [Zamith et al. 2007].

In the parallel programming models, it is necessary to detect which are shared and non-shared parts, which will be treated differently. The independent sections compose tasks that are processed in parallel, like the rendering task. The shared sections, like the GPGPU and the update stages, need to be synchronized in order to guarantee mutual-exclusive access to shared data and to preserve task execution ordering.

Although the threads run independently from each other, it is necessary to ensure the execution order of some tasks that have processing dependence. The first stage that should be run is the update, followed by GPGPU stage, while the render stage runs in parallel to the previous ones. The update and GPGPU stages are responsible for defining the new game state. For example, the former calculates the collision response for some objects, whereas the latter defines new positions for the objects. The render stage presents the results of current game state.

The processing dependence of shared objects needs to use a synchronization object, as applications that use many threads do. Multithread programming is a complex subject, because the tasks in the application run alternately or simultaneously, but not linearly.

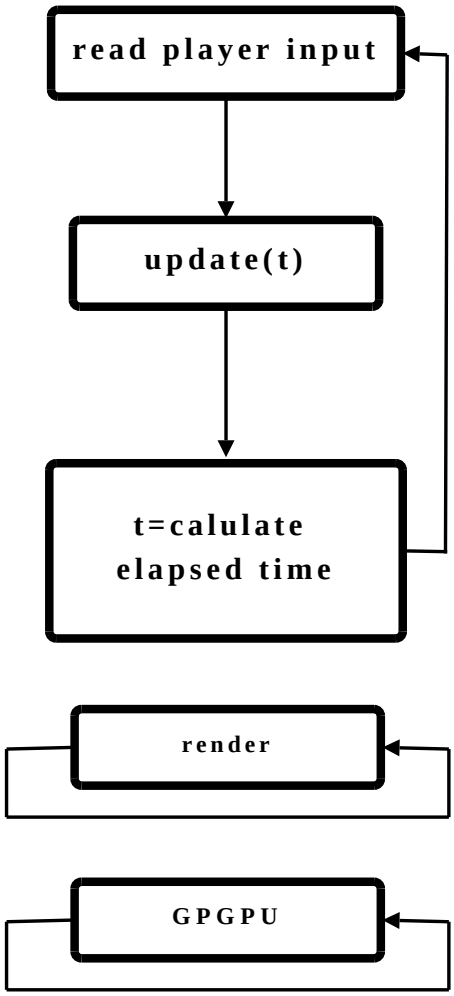


Figure 3: Multithread uncoupled with GPGPU model

Hence, synchronization objects are tools for handling task dependence and execution ordering. This measure should also be carefully applied in order to avoid thread starvation and deadlocks. The current implementation of the presented model uses semaphores as synchronization object.

The framework architecture provides the abstract class AbstractAppRunner to run a generic application. The methods in that class represent game loop stage and other system events, that are dispatched during the application life cycle. The approach presented in this work implements the concrete class GPGPUAppRunner, derived from AbstractAppRunner, which is a composed by a main loop and two thread objects.

The thread objects are instances of two concrete classes: GameLoopThread and GPGPULoopThread. The GPGPULoopThread, which extends the abstract class AbstractThread, defines virtual methods to run the thread loop and to change the semaphores. The GPGPURunner loop runs the data input and render stages, the GameLoopThread loop runs the update stage, and the GPGPULoopThread loop runs the GPGPU stage. Figure 7 shows the UML class diagram of this architecture.

Even if the tasks should obey a predefined execution ordering, the multithread approach makes it possible to run the stages simultaneously.

The distribution model presented in this work is fixed, being the distribution of the tasks between the CPU and GPU defined previously. The implementation of a desirable automatic model of process distribution is a very intricate issue and even more if it is considered distribution between two different hardware architectures: the conventional CPU and the GPU. Besides, as it was mentioned previously, not all kinds of tasks can be processed by the GPU.

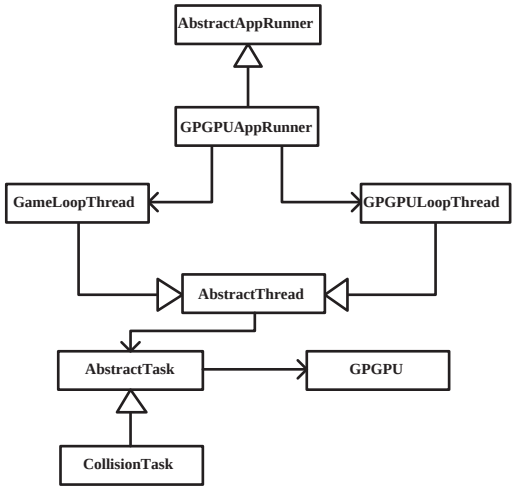


Figure 4: Main classes of the proposed model

5 Example and Results

The case study implements collision detection among moving solids. In a previous version, the collision detection process was implemented entirely on CPU, in the updated stage. The subsequent version shows how this process can be performed in the GPGPU stage.

The use of GPU for general-purpose computation requires an OpenGL extension called FBO (frame buffer object). A FBO is configured from parameters related to the storage format of textures in the GPU, the floating point precision (the floating point precision is 32bit IEEE standard), and the texel components (RGBA color channels). The concrete class GPGPU encapsulates details regarding the FBO setup and the fragment shader program. This class defines and implements methods that help complex manipulation of resources, like automatic configuration, texture transfering (uploading to GPU memory and downloading from frame buffer), and shader running.

As the GPGPU stage run tasks, it is necessary an intermediate layer where the tasks will be implemented. Therefore, an intermediate layer is built between the application (the game) and the GPGPU class. The objective of the class GPGPU is to create a communication interface between the application and the GPU, allowing tasks to be scheduled either for the CPU or for the GPU, with no interference from the developer. To represent the layer, the class AbstractTask was defined. This one holds a GPGPU object which in turn extends from the GPGPU class. The collision task, which extends AbstrackTask, was defined for implementing collision among solid objects.

Two methods of the class AbstractTask deserve special attention: execCPU and execGPU. The former is where the CPU algorithm will be implemented, and in the latter is where the GPU algorithm will be implemented. Therefore, for each task that will be processed on GPU is necessary to write a new class that extends the class AbstractTask. AbstractThread is an abstract class that implements APIs for thread creation and manipulation. The class declares a pointer to an object of a class derived from AbstractTask. GameLoopThread and GPGPULoopThread are extensions of AbstractThread; thus, GameLoopThread runs tasks on CPU by invoking execCPU task method, and GPGPULoopThread runs tasks on GPU by invoking execGPU task method (see UML class diagram in Figure 7).

The class AbstrackTask holds a vector of pointers to solid objects. The method execGPU is responsible for mapping some object properties to floating point RGBA textures. The properties are position, velocity, and acceleration. As these properties have three dimensions, the execGPU method uses three textures, one for each property. Each value in the property is mapped to a texel in the texture, so the RGB values store those values. Then, the class GPGPU uploads these textures to the GPU memory. As soon as the textures

are uploaded, the GPU will run the shader program $(n-1)$ times, where n is the number of solid objects in scene. The number of collision tests generated is

$$C_n^2 = \binom{n}{2} \quad (3)$$

After the shader program runs, the GPGPU class assembles the results in a output texture. A texel in this texture corresponds to an object that was tested. For each texel, the RGB values store the corrected object position and the alpha value indicates if a collision was found (one) or not (zero).

For example, consider a scene with sixteen objects where their positions are mapped in a texture. Figure 5 illustrates a simple representation of how shader variables (handles) are used to access texels, where each object position is mapped to a color. The first handle points to the first texel of the texture. The second handle makes it possible to access neighbouring texels by offsetting the first handle. Therefore, it is possible to access different solid objects, and then to perform collision detection among them.

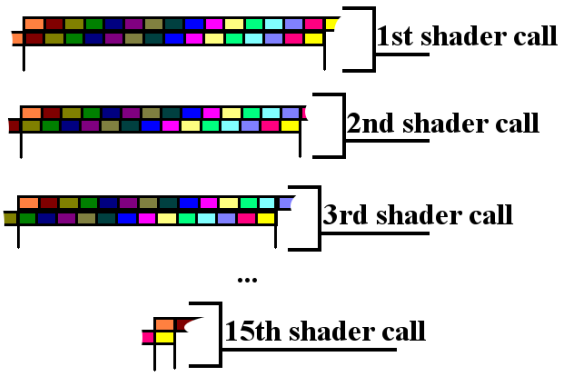


Figure 5: Representation of objects position in texture.

The parallelism between positioning camera and collision detection happens at a distinct moment. In this case, there is not parallelism among rendering, object update, and collision detection. The moving objects represent the part of application that shares resources.

In the case study, there is parallelism among player input update and object rendering, and camera positioning. Figure 6 shows which tasks are executed simultaneously and which are executed lineally. So, the render stage and camera update have even happened, but on the other hand, collision and update task have never happened together. The alternation is guaranteed by semaphor signals between the threads, that is, the signals are realized between the threads (GameLoopThread and GPGPULoopThread) that execute the updated and GPGPU stages (collision task). The synchronization is guaranteed by two variables: RedSync and GreenSync, whose values are shared, i.e., the RedSync variable of one thread shared the same value that the GreenSync variable of the other thread. Thus, the tasks are exclusively executed. After time t5 a new iteration is executed. The figure also shows how tasks are distributed between CPU and GPU: main thread runs on CPU and is responsible for camera and object update; render and GPGPU threads runs on GPU and are responsible for scene rendering and GPGPU collision detection.

Figure 7 illustrates two colliding objects. At time t0, both bodies are about to collide, and at time t1 they enter a collision state, at this moment the shader program is invoked by GPGPU thread. The shader program running on the GPU detects this collision and corrects the object positions. At time t2, their positions are updated because they are read from the FBO.

In order to evaluate the proposed model, its performance was compared with the original loop model of the framework [Valente et al. 2005]. The hardware used was a Pentium D 3.4Ghz, 1GB of memory, and a GPU NVIDIA GeForce 6200 AGP.

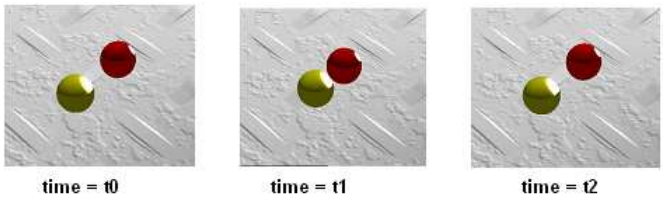


Figure 7: Collision between two spheres.

Two classes of tests were constructed using the framework. The first one uses the single coupled synchronized model and the other the multithread uncoupled with GPGPU, as proposed in this work. For each class, 10 tests were executed to measure the processing time in the GPU and the CPU. Each class consists in 500 collisions between 16 moving solids positioned in the scene in an arbitrary way, without any user interaction.

Tables 1 and 2 show the results corresponding to the multithread with GPGPU and simple coupled synchronized models, respectively. Column labeled **total col. t.** represents the time for computing the 500 collisions, **total app. t.** is the total running time of the application, **work** is the effective time in the GPU and CPU (all of them in seconds), and **FPS** the number of frames per second.

Table 1: Results: multithread uncoupled w. GPGPU model

total col. t.	total app. t.	works	FPS
1,578	0,015	0,030000014	77,3131
1,938	0,016	0,047000030	79,9794
1,703	0,016	0,062999996	76,9231
2,219	0,016	0,032000001	84,2722
1,719	0,016	0,032000004	77,9523
2,093	0,016	0,032000085	84,5676
2,234	0,016	0,016000003	83,2587
1,968	0,016	0,045999954	82,8252
1,312	0,016	0,030999969	70,8841
2,203	0,016	0,015999996	83,9764

Table 2: Results: simple synchronized coupled model

total col. t.	total app. t.	works	FPS
22,625	0,016	0,063000096	240,442
83,594	0,016	0,330000447	242,290
44,234	0,016	0,232999674	242,302
27,140	0,016	0,329000170	244,068
42,828	0,016	0,266999744	240,870
50,063	0,016	0,234000313	242,714
14,328	0,016	0,063999999	238,554
15,328	0,016	0,126000026	238,192
30,188	0,016	0,109999970	241,420
22,110	0,016	0,172999781	240,299

Table 3 represents the comparison of the mean processing times corresponding to 10 test instances. Lines labeled **FPS** represents the number of frames per second and **work** the effective processing time, respectively. Lines **time**, **minimal**, and **maximum** correspond respectively to the mean, the minimum, and the maximum processing times of the 10 instances (in seconds).

The results in Tables I, II and III show an increase in performance obtained by using the multithread uncoupled model with GPGPU. This increase in performance is due to the concurrent execution of the tasks.

Using the GPGPU component of the architecture considerably reduced the processing time of the collision detection part of the system. This caused an overall decrease in the processing time of the application. Besides, the tests have shown that the frame rate de-

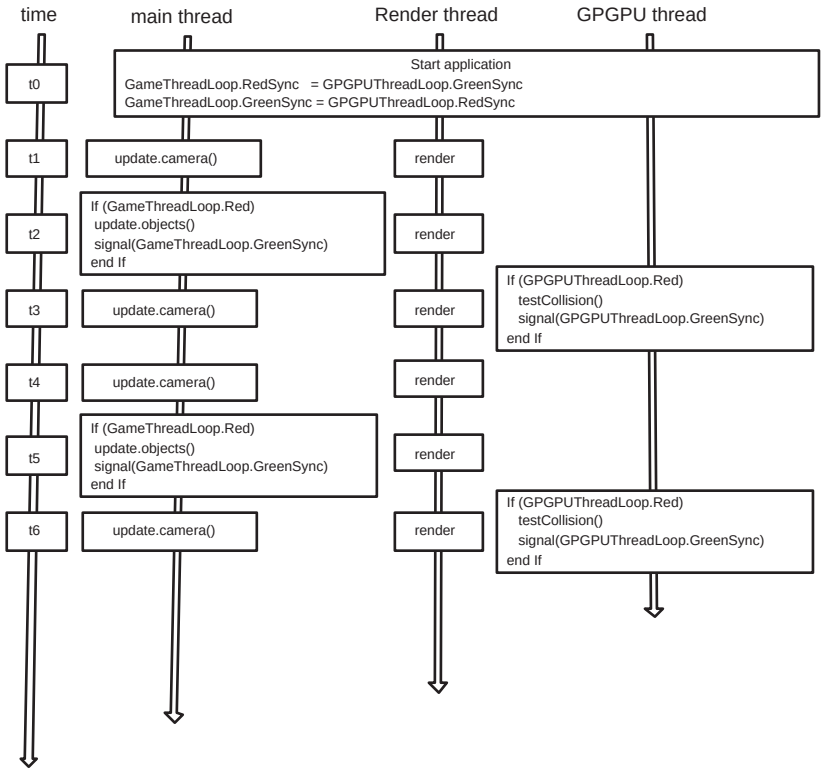


Figure 6: Task parallelism representation

Table 3: GPU and CPU comparatives

	GPU	CPU
FPS	80,1952100	241,1151000
works	0,0345000	0,1929000
time	1,8967000	35,2438000
minimal	0,0160000	0,0630001
maximum	0,0630000	0,3300004

crease, to enable GPGPU processing, did not affect the quality of the animation, which was preserved.

6 Conclusions and Future Work

Balancing the load between CPU and GPU is an interesting approach for achieving a better use of the computational resources in a game or virtual and augmented reality applications. By doing this it is possible to dedicate the additional free computational power to other tasks as AI and complex physics, which could not be done in more rigid or sequential architectures.

This work has demonstrated this possibility by introducing a new stage in a multithread uncoupled game engine architecture, which is responsible for general-purpose processing on the GPU. As it was exemplified by a solution for the collision detection problem, it is possible to use this same approach for other problems and tasks as AI and those listed in Section 2, which can be also processed in this component of the architecture by the GPU. In order to do this it suffices to do the correct mapping of the objects to the appropriate textures and design the corresponding shader responsible for implementing the algorithm associated to the solution.

The results tabled in Section 5 demonstrate that the use of GPU for general-purpose computation is a promising way to increase the performance in game engines, virtual and augmented reality systems, and other similar simulation applications.

In the most recent graphics processors, as the GeForce 8 series, it is possible to use one of its GPUs for running a thread responsible for managing the load balancing between CPU and GPU. The authors intend to pursue this direction in a future work. Another point to be investigated is how to detect in a more automatic way which processes are appropriate for CPU or GPU allocation.

References

BACIU, G., AND WONG, W. S. K. 2003. Image-based techniques in a hybrid collision detector. *IEEE Transactions on Visualization and Computer Graphics 9*, 2, 254–271.

BOLZ, J., FARMER, I., GRISPUN, E., AND SCHRÖDER, P. 2003. Sparse matrix solvers on the GPU: conjugate gradients and multigrid. *ACM Transactions on Graphics 22*, 3, 917–924.

BOND, A. 2006. Havok FX: GPU-accelerated physics for PC games. In *Proceedings of Game Developers Conference 2006*. Available at `www.havok.com/content/view/187/77/`.

EBERLY, D. H. 2004. *Game Physics*. Morgan Kaufmann.

ERICSON, C. 2005. *Real-Time Collision Detection*. Morgan Kaufmann.

FATAHALIAN, K., SUGERMAN, J., AND HANRAHAN, P. 2004. Understanding the efficiency of GPU algorithms for matrix-matrix multiplication. In *Graphics Hardware 2004*, 133–138.

FEIJÓ, B., PAGLIOSA, P. A., AND CLUA, E. W. G. 2006. Visualização, simulação e games. In *Atualizações em Informática*, K. Breitman and R. Anido, Eds. Editora PUC-Rio, 127–185. (In Portuguese).

GALOPPO, N., GOVINDARAJU, N. K., HENSON, M., AND MANOCHA, D. 2005. LU-GPU: efficient algorithms for solving dense linear systems on graphics hardware. In *Proceedings of the ACM/IEEE Conference on Supercomputing*, 3–14.

GOVINDARAJU, N. K., REDON, S., LIN, M. C., AND MANOCHA, D. 2003. CULLIDE: interactive collision detection between complex models in large environments using graphics hardware. In *Graphics Hardware 2003*, 25–32.

GREEN, S., 2003. NVIDIA cloth sample. Available at `download.developer.nvidia.com/developer/`

SDK/ Individual_Samples/ samples.html# glsl_physics.

HEIDELBERGER, B., TESCHNER, M., AND GROSS, M. 2004. Detection of collisions and self-collisions using image-space techniques. *Journal of WSCG 12*, 3, 145–152.

JAMES, D. L., AND PAI, D. K. 1999. Accurate real time deformable objects. In *Proceedings of ACM SIGGRAPH 99*, 65–72.

KIPFER, P., SEGAL, M., AND WESTERMANN, R. 2004. UberFlow: a GPU-based particle engine. In *Graphics Hardware 2004*, 115–122.

KOLB, A., LATTA, L., AND RESK-SALAMA, C. 2004. Hardware-based simulation and collision detection for large particle systems. In *Graphics Hardware 2004*, 123–132.

KRÜGER, J., AND WESTERMANN, R. 2003. Linear algebra operators for GPU implementation of numerical algorithms. *ACM Transactions on Graphics 22*, 3, 908–916.

OWENS, J. D., LEUBKE, D., GOVINDARAJU, N., HARRIS, M., KRÜGER, J., LEFOHN, A. E., AND PURCELL, T. J. 2007. A survey of general-purpose computation on graphics hardware. *Computer Graphics Forum 26*. To appear.

RUMPF, M., AND STRZODKA, R. 2005. Graphics processor units: New prospects for parallel computing. In *Numerical Solution of Partial Differential Equations on Parallel Computers*, vol. 51 of *Lecture Notes in Computational Science and Engineering*. Springer-Verlag, 89–134.

VALENTE, L., CONCI, A., AND FEIJÓ, B. 2005. Real time game loop models for single-player computer games. In *Proceedings of the IV Brazilian Symposium on Computer Games and Digital Entertainment*, 89–99.

VALENTE, L. 2005. *Guff: um framework para desenvolvimento de jogos*. Master's thesis, Universidade Federal Fluminense. (In Portuguese).

ZAMITH, M. P. M., CLUA, E. W. G., CONCI, A., MOTENEGRO, A., PAGLIOSA, P. A., AND VALENTE, L. 2007. Parallel processing between gpu and cpu: Concepts in a game architecture. *IEEE Computer Society*, 115–120. ISBN: 0-7695-2928-3.

ZELLER, C. 2005. Cloth simulation on the GPU. In *ACM SIGGRAPH 05: ACM SIGGRAPH 2005 Sketches*.

Algoritmos Evolutivos para a produção de NPCs com Comportamentos Adaptativos

Marcio K. Crocomo Mauro Miazaki Eduardo do Valle Simões

Universidade de São Paulo – Instituto de Ciências Matemáticas e de Computação

Figura 1: Tela do jogo – no centro, pode-se visualizar o ambiente com as duas populações NPCs (criaturas verdes e vermelhas), a criatura do jogador (azul) e as plantas; na margem direita e inferior, encontram-se as informações sobre o jogo em andamento.

Resumo

Neste trabalho, foi elaborado um jogo que utiliza um Algoritmo Evolutivo (AE) no qual os comportamentos adotados pelos agentes controlados pelo computador (*Non-Player Characters* – NPCs) se adaptam de acordo com as estratégias utilizadas pelo jogador. Para a elaboração deste jogo, diversos AEs foram testados, com o objetivo de encontrar uma configuração capaz de adaptar as estratégias do computador ao estilo do jogador. Este trabalho vem contribuir com o estado da arte, no qual há poucas publicações envolvendo AEs no aprendizado on-line para jogos (aprendizado durante a interação com o jogador). Alguns autores chegam a afirmar que AEs não são uma alternativa viável para esse tipo de aplicação por não cumprirem o requisito de eficiência. Contrariando essas afirmações, o software produzido resultou num jogo em que os personagens controlados pelo computador conseguem oferecer um grau de adaptabilidade às ações do jogador devido ao AE implementado. Além disso, foi possível observar o surgimento de estratégias originais muito eficientes utilizadas pelo computador que superaram as expectativas dos desenvolvedores.

Palavras Chave: Inteligência Artificial, Computação Evolutiva, Vida Artificial, Aprendizado on-line, Adaptação Direta.

Contatos dos autores:
{marciokc,maurom}@gmail.com.br
simões@icmc.usp.br

1. Introdução

A evolução dos jogos eletrônicos nos últimos anos vem gerando uma demanda por sistemas computacionais de alto desempenho. Inicialmente voltados ao realismo da ambientação gráfica, fabricantes concentram-se hoje no aprimoramento da inteligência artificial dos personagens não controlados pelo jogador. Técnicas de vida artificial [Mitchell and Forrest 1994] vêm sendo combinadas com atitudes individuais cada vez mais realistas dos personagens para gerar ambientes criativos e surpreendentes para o jogador. Alguns jogos de última geração, como o SPORE[1], da Eletronic Arts, já apostam na evolução gradual dos personagens, em que a seleção "natural" dos mais adaptados é o que

[1] Eletronic Arts, SPORE, http://www.spore.com, acessado em agosto/2006.

define o sucesso das populações controladas pelos jogadores.

Até o momento, os desenvolvedores de jogos têm concentrado maiores esforços na área de computação gráfica. No entanto, vários jogos têm se destacado pela qualidade de sua Inteligência Artificial, como o Black & White 2 e o The Sims. Sweetser [2002] afirma que o próximo foco a ser desvendado no mercado de jogos será com personagens que se comportam de forma realista e podem aprender e se adaptar, ao invés de personagens com maiores resoluções e jogos com mais quadros por segundo.

Este trabalho traz contribuições para o campo de Inteligência Artificial aplicada a jogos ao elaborar um jogo que utiliza um Algoritmo Evolutivo (AE) e propor mecanismos evolutivos para o desenvolvimento e otimização de estratégias de ações para personagens não controlados pelo jogador. No meio acadêmico, AEs vêm sendo explorados com grande freqüência, sendo que novas técnicas são constantemente elaboradas. Já na aplicação em jogos comerciais, AEs encontram certa resistência devido ao fato de pesquisadores apontarem a técnica como lenta e capaz de permitir a obtenção de desempenhos ruins [Sweetser 2002; Spronck 2004]. Este trabalho procura verificar tais afirmações e demonstrar que diversas técnicas relacionadas a AE desenvolvidas no meio científico podem ser aplicadas em jogos na prática.

2. Inteligência Artificial em Jogos

Existem muitas técnicas de Inteligência Artificial utilizadas nos jogos eletrônicos atuais. Sweetser [2002], em seu trabalho, cita algumas dessas técnicas, explicando como as mesmas se encontram inseridas no cenário de produção de jogos.

Atualmente, para programar as estratégias utilizadas pelo computador, Máquinas de Estados Finitos [Piccione and Rubinstein 1993] são utilizadas em jogos com uma freqüência maior do que outras técnicas de Inteligência Artificial [Sweetser 2002]. Embora sejam de fácil implementação e geralmente suficientes para o propósito do jogo, essa técnica torna bastante previsível a estratégia utilizada pelo computador [Ponsem 2004]. Exemplos de jogos comerciais que utilizam essa técnica são: Age of Empires, Enemy Nations, Half-Life, Doom e Quake.

Scriptings [Prechelt 2003] são linguagens de programação criadas para simplificar a programação de tarefas complexas para um programa particular. São utilizadas em jogos para permitir que os jogadores possam programar o comportamento dos personagens controlados pelo computador, como no jogo Baldur's Gate, por exemplo. O jogo Neverwinter Nights é outro exemplo de jogo que utiliza scripting.

O papel da Lógica Fuzzy [Elkan 2001] nos jogos atuais geralmente não ultrapassa complexos comandos condicionais (comandos do tipo se-então-senão). Segundo Sweetser, isto se deve à complexidade de se criar sistemas de Lógica Fuzzy. Esta técnica se mostra útil em tomadas de decisão e seleção de comportamentos em sistemas de jogos. Pode-se, por exemplo, utilizá-la para determinar o quão "assustado" um personagem está em relação ao jogador. Os jogos BattleCruiser: 3000AD, Platoon Leader e SWAT 2 são exemplos de jogos comerciais que utilizam Lógica Fuzzy.

A técnica de Flocking [Reynolds 1987] possui como objetivo simular comportamentos naturais de grupos de entidades, como, por exemplo, um rebanho de ovelhas. Essa técnica foi utilizada com sucesso em uma variedade de jogos comerciais criando ambientes realistas para o jogador explorar. Exemplos de jogos que utilizaram essa técnica são: Half-Life, Unreal, e Enemy Nations.

Redes Neurais Artificiais (RNA) [Haykin 2001] podem ser utilizadas para ensinar o computador a imitar o comportamento do jogador. Embora Sweetser diga que Redes Neurais Artificiais terão um grande papel a desempenhar nos jogos comerciais em um futuro próximo, os desenvolvedores de jogos têm sido relutantes em aceitar o uso dessa técnica, devido ao fato de permitirem gerar comportamentos "inadequados" para os personagens controlados pelo computador. Essa deficiência é também apontada em outros algoritmos de aprendizado, como, por exemplo, os Algoritmos Evolutivos. No entanto, os jogos que empregaram RNAs não a utilizaram no aprendizado durante o jogo (aprendizado on-line), mas sim no treinamento dos agentes do jogo durante o desenvolvimento do mesmo (aprendizado off-line). Exemplos de jogos que utilizam Redes Neurais: Black & White, Creatures, Dirt Track Racing e Heavy Gear.

A Inteligência Artificial em jogos utiliza técnicas de aprendizado de máquina que podem ser utilizadas on-line ou off-line [Yannakakis 2005]:

- No aprendizado off-line, as modificações se dão durante a produção do jogo, quando as estratégias utilizadas se adaptam jogando contra outras técnicas programadas; ou seja, sem a intervenção de um jogador humano.

- No aprendizado on-line, a adaptação do jogo ocorre durante a interação com o usuário, ou seja, depois que o jogo já foi produzido. Para que o aprendizado on-line funcione na prática, precisa-se que seja rápido, efetivo, robusto e eficiente [Spronck 2004].

Em relação a como o aprendizado é realizado, Manslow [2002] explica a diferença entre duas abordagens distintas:

- Adaptação Indireta: Ocorre quando alterações são feitas a certos aspectos do comportamento, baseadas nas estatísticas do jogo. A escolha de quais estatísticas são extraídas, e suas interpretações na forma de mudanças no comportamento dos agentes do jogo, são feitas pelo programador. O papel desse mecanismo de

aprendizado se restringe a extrair informação do jogo e não age diretamente na mudança do comportamento dos agentes. Um exemplo de adaptação indireta é a mudança dinâmica do nível de dificuldade: as informações extraídas do jogo são baseadas no nível das habilidades do jogador; em resposta a isso, é atribuído ao jogo um nível de dificuldade compatível às habilidades do usuário.

• Adaptação Direta: utiliza algoritmos de otimização e de aprendizado por reforço para alterar diretamente o comportamento da IA nas bases que afetam seu desempenho. O algoritmo de aprendizado procura por parâmetros que oferecem o melhor desempenho, isto é, o melhor comportamento para os agentes controlados pelo jogo. A Adaptação Direta geralmente é ineficiente e difícil de controlar. Além disso, é difícil encontrar uma medida adequada de desempenho. No entanto, a adaptação direta apresenta a vantagem de não limitar os comportamentos dos agentes do jogo e requer pouco conhecimento prévio do programador sobre bons comportamentos.

3. Algoritmos Evolutivos em Jogos

Inspirados na Teoria da Evolução de Darwin [Darwin 1998], os Algoritmos Evolutivos (AEs) [Holland 1962] abstraem mecanismos da biologia, tais como o crossover e a mutação, e os aplicam em problemas computacionais com o objetivo de encontrar uma solução adequada [Tomassini 1995]. É possível explicar um AE como sendo um procedimento iterativo. Inicialmente, é gerada de forma aleatória uma população de indivíduos, em que cada um representa uma possível solução para o problema em questão. Cada solução é avaliada por uma função de pontuação, recebendo uma "nota" (fitness score) que indica o quão eficaz é a solução [Tomassini 1995]. Baseando-se nessa pontuação, é selecionado um grupo de indivíduos. Existem vários métodos para essa seleção, como o da Roleta e o Elitismo [Bramlette 1991]. Em seguida, aplicando os operadores de crossover e mutação, uma nova geração é criada (um novo conjunto de soluções). Após isso, é reiniciado o processo a partir da fase de avaliação. Geração após geração, existe uma tendência a se obter resultados cada vez melhores [Mitchell and Forrest 1994]. Caso o problema seja modificado durante a execução do AE, é esperado que este comece a buscar dinamicamente uma nova solução, gerando soluções adaptativas [Watson et al. 2002].

Além das funções de mutação e crossover, outros operadores evolutivos podem ser utilizados, como Funções de Genocídio [Simões 1999], por exemplo. Nesse passo, algumas técnicas especiais foram introduzidas por pesquisadores, conseguindo aumentar o desempenho do AE. Entre elas, a de Hereditariedade [Simões 2000] e Funções de Predação [Crocomo et al. 2004] merecem destaque. Na técnica de Hereditariedade [Simões 2000], ao invés da pontuação de cada indivíduo na geração corrente, o valor utilizado para a seleção do indivíduo mais adaptado, que irá cruzar com todos os outros da população, é o valor médio da pontuação do indivíduo nas últimas cinco gerações. Foram propostos em [Crocomo et al. 2004] e [Simões and Barone 2003] dois tipos de predação: Aleatória e Síntese. Na Predação Aleatória, a cada vinte gerações, o cromossomo do pior indivíduo é selecionado e substituído por um cromossomo aleatório. Na Predação por Síntese, o cromossomo do pior indivíduo é substituído por um cromossomo que tem cada um de seus genes substituído pelo valor mais comum encontrado na população em cada lócus.

Os AEs têm sido bastante utilizados para evoluir estratégias de jogos desde os anos 80. Pode-se citar como exemplo o jogo "Iterated Prisioner's Dilemma" [Yao and Darwen 2000], de grande importância para economistas, cientistas sociais e cientistas da computação. Esse jogo utiliza aprendizado off-line.

Jogos são um bom ambiente de testes para Algoritmos Evolutivos, pois apresentam problemas de co-evolução e evolução multiobjetivo [Koza 1992]. Existem jogos que são criados com o propósito de ser um ambiente de testes para AEs, como é o caso, por exemplo, do jogo "Cellz" [Lucas 2004]. Esses jogos envolvem interação entre os personagens controlados pelos jogadores e os diversos caracteres autônomos, que necessitam de percepção, cognição e determinado grau de adaptabilidade às reações dos jogadores e às variações do ambiente de jogo.

Embora Algoritmos Evolutivos tenham sido explorados intensivamente no meio acadêmico, ainda não foram bem aceitos no desenvolvimento de jogos. Essa resistência vem do fato de alguns pesquisadores apontarem essa técnica como sendo lenta e não eficaz, no sentido de permitir que desempenhos ruins sejam gerados [Sweetser 2002; Spronck 2004]. No entanto, Sweetser reconhece que AEs oferecem oportunidades para desenvolver estratégias em áreas nas quais técnicas tradicionais de IA para jogos apresentam deficiências.

Exemplos de jogos de computador que utilizam Algoritmos Evolutivos são: Creatures; Cloack, Dagger and DNA; e Return to Fire II.

4. Trabalhos Relacionados

Em [Ponsem 2004], Ponsen utiliza Algoritmos Evolutivos para auxiliar na criação de estratégias a serem utilizadas pelo computador. No trabalho realizado por Ponsen, foi utilizado um Algoritmo Evolutivo no aprendizado off-line de jogos para desenvolver bons scripts a serem utilizados em um jogo de estratégia em tempo real. Para tanto, foi utilizado o jogo Wargus[2], um clone do famoso jogo de estratégia Warcraft II. Ponsen utilizou AEs no

[2] Wargus, `http://wargus.sourceforge.net`, acessado em agosto/2006.

aprendizado off-line do jogo, deixando o aprendizado on-line a cargo da técnica Dynamic Scripting, desenvolvida por Spronck [2004].

Spronck afirma em [Spronck 2004] que Algoritmos Evolutivos não são candidatos a serem utilizados no aprendizado on-line em jogos, pois não satisfazem dois dos quatro requisitos apresentados pelo autor: rápido, efetivo, robusto e eficiente. Spronck explica estes requisitos da seguinte forma:

- Rápido: como o aprendizado on-line ocorre durante a execução do jogo, o algoritmo de aprendizado deve ser computacionalmente rápido, para que não atrapalhe o andamento do jogo.
- Efetivo: os scripts adaptados devem ser pelo menos tão desafiadores quanto aqueles programados explicitamente. Conseqüentemente, o mecanismo de aprendizado precisa garantir a geração de uma IA mais eficiente. Esse requisito, aponta Spronck, exclui métodos de aprendizado aleatório, como os Algoritmos Evolutivos, pois podem produzir estratégias muito ruins para serem testadas pelo jogador, prejudicando a jogabilidade.
- Robusto: o mecanismo de aprendizado precisa ser capaz de lidar com uma quantidade de aleatoriedade significante, inerente à maioria dos mecanismos usados em jogos comerciais.
- Eficiente: o processo de aprendizado deve ser eficiente, funcionando com apenas um pequeno conjunto de testes. Spronck acredita que este requisito exclui técnicas de aprendizado lento, como Redes Neurais, Algoritmos Evolutivos e Aprendizado por Reforço, que tomariam muito tempo de jogadores humanos a fim de otimizar suas estratégias.

A técnica Dynamic Scripting desenvolvida por Spronck foi testada utilizando um simulador muito semelhante ao jogo de RPG Baldur's Gate, em que um grupo de personagens controlados pelo computador confronta outro grupo controlado pelo jogador. No trabalho desenvolvido por Yannakakis [2005], um jogo utilizando o modelo predador-presa foi desenvolvido aplicando um AE no aprendizado on-line de agentes controlados pelo computador através de uma Rede Neural. Segundo Yannakakis, os desafios encontrados ao se projetar um mecanismo de aprendizado para oponentes em tempo real são:

- Desempenho em tempo real: o aprendizado deve ter um bom desempenho em tempo real, sendo que geralmente uma única CPU é disponibilizada para a maioria dos jogos. O autor lembra também que a maioria dos jogos comerciais utiliza acima de 90% de sua CPU apenas para engines gráficas.
- Realismo: Yannakakis diz que evolução neural apresenta o potencial para a geração de oponentes adaptativos, mas também pode gerar comportamentos não realistas. Na maioria dos jogos de computador, o jogador interage com um pequeno número de oponentes. Dessa forma, um comportamento não realista é facilmente notado e pode levar a um baixo entretenimento do jogador.
- Adaptação Rápida: um bom aprendizado on-line deve se adaptar rapidamente às mudanças de estratégia feitas pelo jogador. De outra forma, o jogo se tornaria entediante. Este é um grande desafio para o desenvolvedor do mecanismo responsável pelo aprendizado on-line dado os recursos computacionais disponíveis, o comportamento realista e o pequeno número de oponentes que interagem com o jogador.

Yannakakis desenvolveu um conjunto de métricas de entretenimento para medir o interesse dos jogadores em jogos do estilo predador-presa. Nessas métricas, o autor procura medir o desafio do jogo (nível de dificuldade), diversidade de comportamento nos oponentes e se o comportamento dos oponentes predadores é agressivo. O último item é explicado pelo autor como a capacidade dos oponentes predadores se movimentarem e cobrirem uniformemente o cenário. Isso dá a impressão ao jogador que uma estratégia inteligente foi adotada pelo computador.

Sweetser [2002] realizou uma revisão de IA em jogos, comparando várias técnicas como Algoritmos Evolutivos, Árvores de Decisão, Redes Neurais, Máquinas de Estados Finitos, Scripting, Lógica Fuzzy, e Flocking. Ele levantou para cada um desses métodos suas vantagens, desvantagens e aplicações, além de citar jogos que utilizaram essas técnicas. Segundo Sweetser, embora as possíveis aplicações de AEs sejam imensas, eles ainda não foram totalmente aceitos no desenvolvimento de jogos.

5. Análise Crítica

Ponsen utiliza um Algoritmo Evolutivo no aprendizado off-line em jogos, deixando o aprendizado on-line do jogo a cargo da técnica Dynamic Scripting desenvolvida por Spronck e seu grupo. Em seu trabalho, Spronck diz que o aprendizado on-line precisa ser eficiente (aprendendo em um pequeno número de tentativas) e, portanto, técnicas como Redes Neurais Artificiais, Algoritmos Evolutivos e Aprendizado por Reforço [Sutton and Barto 1998] não são opções viáveis.

Embora vários trabalhos envolvendo o uso de Algoritmos Evolutivos no desenvolvimento de jogos tenham sido desenvolvidos, pouco material é encontrado sobre sua utilização em jogos com aprendizado on-line. Spronck afirma que AEs não são uma alternativa viável a ser utilizada no aprendizado on-line em jogos comerciais e explica os motivos que o levam a afirmar isso. No entanto, suas afirmações ainda não foram de fato demonstradas.

Diferente do trabalho de Ponsen [2004], que utilizou um AE no aprendizado off-line, e contrariando as afirmações de Spronck, o jogo desenvolvido neste trabalho utiliza um AE no aprendizado on-line. Como

em pouco tempo de jogo é possível notar uma adaptação dos agentes controlados pelo computador (aprendendo a fugir do jogador e a ir em direção às plantas para se alimentar, por exemplo), a afirmação feita por Spronck sobre a viabilidade de se utilizar AE, por não serem eficientes, é colocada em dúvida.

No requisito "Eficiência", descrito por Spronck, os scripts adaptados um jogo pode ser o de selecionar o script mais adequado contra a estratégia adotada pelo jogador, dentre um conjunto de scripts previamente programados. Isso mostra que AEs também podem ser utilizados no aprendizado on-line. No entanto, as qualidades (rápido, efetivo, robusto e eficiente) apontadas por Spronck para um algoritmo de aprendizado on-line, são importantes critérios de avaliação para esses algoritmos, já que foram aceitos pela comunidade científica como métricas de qualidade para aprendizado on-line.

O trabalho de Yannakakis, que também coloca em dúvida as afirmações de Spronck, utiliza um AE no aprendizado on-line de um jogo no modelo predador presa. Yannakakis diz que um problema enfrentado pelo seu trabalho é o realismo apresentado pelo jogo, pois AEs podem gerar comportamentos vistos pelos jogadores como sendo não realistas. Esse problema pode ser resolvido fazendo uso da idéia proposta por Spronck, de utilizar um conjunto de scripts previamente programados, contendo, cada um, comportamentos realistas. Ao se utilizar um AE para selecionar o script mais adequado para enfrentar a estratégia do jogador, é garantido que os comportamentos adotados pelos personagens irão apresentar o realismo desejado.

6. Software Desenvolvido

O Ecossistema Artificial desenvolvido e suas funções evolutivas foram implementados em linguagem C++. Nesse ambiente, os seres virtuais devem se adaptar e "sobreviver" na forma de criaturas carnívoras, herbívoras ou onívoras. O ambiente inclui funções que permitam que os indivíduos selecionem parceiros de acordo com seu desempenho no ambiente para se reproduzir e renovar a população. O funcionamento dos mesmos encontra-se explicado a seguir.

6.1 O Jogo Desenvolvido

O jogo criado consiste de um Ecossistema Artificial contendo plantas e três tipos de criaturas: azuis, verdes e vermelhas. A interface do jogo pode ser vista na Figura 1. Cada criatura possui os seguintes parâmetros:

• "Energia": inicia em 1000 e, com o passar do tempo, vai sendo incrementado. A "Velocidade" também influencia na "Energia". Quanto mais rápida a "Velocidade", mais rápido é o decremento deste parâmetro, pois maior é o gasto de energia. A comida também influencia, mas de forma inversa, aumentando o nível de "Energia". Quando o valor atinge 0, o indivíduo morre. Este parâmetro não é controlado pelo AE.

• "Herbcarn": o valor deste parâmetro varia no intervalo [0,6] e indica se a "criatura" é carnívora ("Herbcarn" = 0), herbívora ("Herbcarn" = 6) ou onívora (valor entre 0 e 6). Este parâmetro também define o nível de eficácia de cada tipo de alimento (planta ou carne) em aumentar a "Energia", de acordo com as equações:

$Energia = Energia + Herbcarn * 100$ (para plantas)
$Energia = Energia + (6 - Herbcarn) * 150$ (para carne)

Assim, quando "Herbcarn"=0, o indivíduo só come carne e a cada vez que ele se alimenta a sua "Energia" aumenta 900 (6*150) pontos. Quando "Herbcarn"=6, de forma semelhante, o indivíduo só come plantas e a cada vez que ele se alimenta a sua "Energia" aumenta 600 (6*100) pontos. No caso dos valores intermediários, de 1 a 5, o indivíduo é onívoro, ou seja, come tanto carne quanto planta e o valor somado da "Energia" na ingestão de cada um dos dois tipos de alimentos é calculado pelas equações apresentadas acima. Uma outra influência do parâmetro "Herbcarn" nos indivíduos é que criaturas com "Herbcarn" menor podem se alimentar de criaturas com "Herbcarn" maior. Criaturas com "Herbcarn" de valores iguais ou criaturas de uma mesma população não podem se alimentar umas das outras.

• "Velocidade": varia no intervalo [1,5] e representa o quão rápido é um indivíduo. Em uma perseguição, o mais veloz ganha. Quanto maior o atributo "Velocidade", mais ativo é o indivíduo e, portanto, mais rápido o atributo "Energia" é decrementado a cada turno (ou ação tomada). A equação abaixo regula este fator.

$Energia = Energia - Velocidade$ (ao andar)
$Energia = Energia - (Velocidade * 2)$ (ao correr)

• "Navegador": a ação a ser tomada é indicada por uma tabela de controle de movimentação. A Tabela 1 apresenta um exemplo.

As entradas do "Navegador" são informações sobre o que se encontra perto do indivíduo, dadas pelas quatro primeiras colunas da Tabela I. Se houver uma planta por perto (dentro de um raio de 60 pixels, em um ambiente de 640x480 pixels), a entrada "Planta" se encontra no estado 1, senão, o estado é 0. O mesmo ocorre quando um membro da população 1 ou das outras populações se encontram por perto.

Tabela I: Exemplo de "Navegador" com 16 situações e possíveis saídas para cada situação.

Planta	Pop1	Pop2	Pop3	Saída
0	0	0	0	0
0	0	0	1	9
…				
1	1	1	0	5
1	1	1	1	8

Para cada situação, o “Navegador” possui 10 saídas possíveis, descritas a seguir:

- 0: Andar aleatoriamente para uma direção.
- 1: Correr aleatoriamente para uma direção.
- 2: Correr para o local em que se encontra a planta mais próxima (perseguir).
- 3: Correr para a direção oposta de onde se encontra a planta mais próxima (fugir).
- 4: Correr para o local em que se encontra o membro da população 1 mais próximo (perseguir).
- 5: Correr para a direção oposta de onde se encontra o membro da população 1 mais próximo (fugir).
- 6: Correr para o local em que se encontra o membro da população 2 mais próximo (perseguir).
- 7: Correr para a direção oposta de onde se encontra o membro da população 2 mais próximo (fugir).
- 8: Correr para o local em que se encontra o membro da população 3 mais próximo (perseguir).
- 9: Correr para a direção oposta de onde se encontra o membro da população 3 mais próximo (fugir).

A cada ação de andar tomada pelo indivíduo, este se move 1 unidade do cenário na direção desejada e sua “Energia” diminui no valor de seu parâmetro “Velocidade”. Quando o indivíduo corre, sua “Energia” diminui em 2 x “Velocidade” e ele se move na direção desejada em “Velocidade” unidades.

A tela do jogo é do tamanho 800x600 pixels e nela podem ser identificados os seguintes componentes importantes:

- Mundo: É a área da tela onde o jogo se passa. O retângulo maior, no canto superior esquerdo da tela, possui as dimensões de 640x480 pixels e é o Ecossistema Artificial no qual as criaturas (personagem do jogador e agentes controlados pelo computador) habitam. Cada indivíduo possui resolução de 32x32 pixels. As pequenas árvores espalhadas pelo cenário representam a vegetação, ou seja, o alimento para as criaturas onívoras e herbívoras.
- Criatura azul: É o Personagem controlado pelo jogador.
- Round: o número abaixo da palavra Round (canto superior direito) indica quantas gerações as criaturas controladas pelo computador tiveram para evoluir. Se Round=1, todas as criaturas contêm controladores inicializados aleatoriamente. Conforme Round aumenta, as criaturas tiveram mais gerações para evoluir e, portanto, a dificuldade do jogo tende a aumentar.
- Survived: é o número de vitórias obtidas pelo jogador até o Round atual; o jogador obtém uma vitória quando é a única criatura sobrevivente.
- Killed: é o número de derrotas obtidas. O jogador é derrotado quando é morto antes que a areia do tempo, representado na imagem da ampulheta, termine. Isso pode ocorrer quando a “Energia” do jogador termina (morrendo de fome) ou quando ele é predado por uma criatura carnívora.
- Draw: é o número de empates até o presente momento. Caso a areia da ampulheta termine e, além do jogador, algum personagem do computador ainda se encontre vivo, o resultado da partida é considerado um empate e Draw aumenta em uma unidade.
- Arbustos Consumidos: representado pelo número que se encontra no canto inferior direito da tela, ao lado do arbusto. Apresenta o número de plantas consumidas pelo jogador desde o início do jogo.
- Criaturas Consumidas: abaixo do número anterior, marca o número de criaturas consumidas pelo jogador desde o início do jogo.
- Energy: dado pelo número no canto inferior esquerdo da tela. Representa o total de “Energia” do personagem controlado pelo jogador, iniciada com o valor 1000. Caso o número chegue a 0, o jogador perde o Round.
- Ampulheta: ao lado de Energy, representa quanto tempo falta para acabar o Round atual. Caso o tempo termine, se ainda existam criaturas da população verde (controladas pelo computador) vivas, o Round é considerado empatado.
- Nível de Dificuldade: no canto inferior da tela, ao lado da ampulheta, baseado no fitness dos três melhores indivíduos de cada população, apresenta um valor que estima a dificuldade oferecida atualmente ao jogador. Existe um medidor de dificuldade para cada população controlada pelo computador: verde (medidor da esquerda) e vermelha (medidor da direita).

O jogo pode ser jogado de dois modos, descritos a seguir:

- Modo1: No primeiro modo, o jogador controla uma criatura onívora e possui como objetivo sobreviver até que todas as criaturas herbívoras (população de criaturas verdes) controladas pelo computador morram. Os herbívoros podem morrer predados pelo jogador ou sem energia (falta de alimento).
- Modo2: No segundo modo, além das criaturas herbívoras controladas pelo computador, existem também as carnívoras (população de criaturas vermelhas). O jogador se encontra no papel de um onívoro que deve sobreviver se alimentando e fugindo dos predadores até o próximo Round. A única maneira de uma criatura carnívora morrer é por falta de energia.

Nos dois modos de jogo, o objetivo do jogador (que controla a criatura azul) é o de ser a última criatura sobrevivente. Caso isso ocorra, o jogador obtém uma vitória, isto é, o valor do campo Survived é aumentado em uma unidade. Quando o jogador morre antes que o tempo da partida acabe, ele é derrotado naquele Round

(o campo Killed é aumentado em um). No caso do tempo acabar com o jogador vivo e alguma criatura controlada pelo computador também viva, o Round é considerado empatado (o campo Draw é incremetado).

6.2 As Funções Evolutivas Implementadas

O AE utilizado tem como papel adaptar o "Navegador" de cada população. No jogo implementado, o parâmetro "Velocidade" é fixado em 3 para qualquer personagem e os valores de "Herbcarn" são:

- Personagem do jogador (azul): 2
- Criaturas Verdes: 6
- Criaturas Vermelhas: 0

O cromossomo a ser evoluído pelo AE é a saída do navegador (coluna mais a direita da Tabela I), codificado como um vetor de inteiros. Essas saídas definem os comportamentos apresentados pelas criaturas no ambiente.

Figura 2: Fluxograma do Algoritmo Evolutivo

A Figura 2 ilustra os passos seguidos pelo AE. Inicialmente, os parâmetros dos cromossomos são gerados aleatoriamente. As populações são testadas no ambiente junto com o personagem do jogador até que a partida termine. Para cada população controlada pelo computador, o membro que sobrevive por mais tempo (medido em turnos ou ações tomadas) é selecionado para se tornar o "pai" da próxima geração. Caso mais de um indivíduo sobreviva pelo mesmo tempo, o atributo "Energia" serve como critério de desempate, sendo selecionado o indivíduo com maior "Energia". Aplicam-se, em seguida, as funções de crossover e mutação para se gerar a próxima geração.

Na etapa de Crossover, os cromossomos de todos os outros membros restantes da população, com exceção do "pai", possuem 50% de chance de ter cada um dos seus parâmetros substituídos pelos do "pai". Na Mutação, cada parâmetro do cromossomo apresenta 2% de chance de ser substituído por um valor aleatório dentro do intervalo apropriado.

O AE tem como objetivo, a cada partida, adaptar as estratégias tomadas pelas populações controladas pelo computador em relação às estratégias adotadas pelo jogador durante o jogo. Para tornar o AE mais eficiente, além da configuração tradicional que utiliza apenas os operadores de mutação e crossover, foram aplicados outros operadores evolutivos:

1) Hereditariedade [Simões 2000]: utiliza a média nas últimas cinco gerações do fitness de cada indivíduo para seleção do cromossomo "pai".

2) Predação Randômica [Simões and Barone 2003]: substitui a cada 20 gerações o cromossomo de menor pontuação por um novo cromossomo aleatório, visando inserir diversidade na população.

3) Predação por Síntese [Crocomo et al. 2004]: substitui a cada geração o cromossomo de menor pontuação (que não seja o último a sofrer predação por síntese) por um novo cromossomo, sendo cada parâmetro substituído pela moda (o valor mais encontrado entre os membros da população) daquele parâmetro nos outros cromossomos da população.

7. Experimentos Realizados

O jogo foi utilizado como uma plataforma de testes para encontrar o AE mais eficiente, fazendo com que duas populações de criaturas disputassem entre si o domínio do ambiente. Cada uma dessas populações foi evoluída por uma instância diferente do AE, independentes entre si. Dessa forma, o AE controlando a população que dominou o ambiente na maioria das gerações foi considerado o mais eficiente.

No primeiro experimento realizado, considerou-se para determinar qual população dominou a outra em uma determinada geração, o critério: qual população possui o indivíduo com o maior fitness. Dizemos que a população A dominou o ambiente na geração i se nesta geração o melhor fitness da população A foi superior ao melhor fitness da população B. Assim, a taxa de dominação do ambiente é:

$$txdom(X) = \frac{f(X)}{NG}$$

onde:

X	População A ou B
$txdom(X)$	Taxa de dominação do ambiente
$f(X)$	Número de gerações em que a população X dominou o ambiente
NG	Número de gerações transcorridas

Com isso, para determinar se a população A dominou a B ou o inverso, pode-se calcular:

$$dom(A,B) = txdom(A) - txdom(B)$$

Dessa forma, podem-se obter três casos:

1. Se $dom(A,B) < 0$, então a População B dominou o ambiente na maioria das gerações.
2. Se $dom(A,B) = 0$, então houve empate.
3. Se $dom(A,B) > 0$, então a População A dominou o ambiente na maioria das gerações.

Como o ambiente proposto é dinamicamente mutável e, portanto, um ambiente com muito ruído, foi necessário adotar um rigoroso critério estatístico para poder afirmar a superioridade de um AE em relação ao

outro. Por exemplo, como saber se a diferença de 4% de dominação do ambiente é suficiente para afirmar a superioridade do AE da população A em relação ao AE da população B? Portanto, para tornar confiáveis os dados colhidos, foi utilizado o cálculo do erro máximo de estimação, com um nível de confiança de 95% [Francisco 1993]. O seu cálculo é realizado a partir dos valores amostrados até a iteração corrente e é definido pela equação:

$$erro = \sqrt{\frac{s^2}{NC}}$$

onde:

erro	Erro máximo de estimação
s^2	Variância amostral
NC	Nº de competições realizadas (amostras)

A Figura 3 exibe o algoritmo implementado, já incorporando *erro* e *f*(X) (internamente, na função *dom*(*A*,*B*)) para solucionar os problemas anteriormente citados. A cada iteração do laço, é realizada uma competição. *NC* registra o número de competições realizadas. A competição realizada no passo 4 é igual ao realizado no experimento anterior. Em cada competição realizada, os cromossomos, gerados de forma aleatória, são iguais para as duas populações, A e B, mas diferem de uma competição para a outra. No passo 5, é calculado qual população, A ou B, dominou o jogo ou se houve empate, em cada geração. O cálculo do *erro*, realizado no passo 6, considera os resultados das competições realizadas até a competição atual *NC* como suas amostras. As iterações continuam até que: o número máximo de competições permitido (200) seja alcançado; ou a condição de erro máximo de estimação (5%) mais o número mínimo de competições permitido (30) seja atingido.

Os dados coletados dessa forma permitem verificar a superioridade de um AE sobre o outro, levando em consideração o erro máximo obtido. Foram então realizados experimentos comparando AEs utilizando as técnicas apresentadas na seção 6.2 (Hereditariedade e Predações) com um AE tradicional (que utiliza apenas crossover e mutação).

```
1  NC = 0;
2  Repita
  3 NC = NC + 1;
  4 Realize uma competição de
    até 1000 gerações;
  5 Calcule txdom(B);
  6 Calcule o erro;
7 Até que (NC > 200) ou ((NC > 30)
      e (erro < 0.05 * txdom_média(B)))
```

Figura 3: Algoritmo utilizado para a coleta dos dados

Após a realização desses experimentos, foi possível selecionar um AE para evoluir o comportamento do computador no jogo elaborado. O experimento realizado permitiu observar os comportamentos que emergiram na população controlada pelo computador, em meio a jogos contra um jogador humano.

8. Resultados Obtidos

Os dados coletados pelo experimento para comparação dos AEs podem ser observados na Tabela 2. A população A utiliza sempre um AE tradicional e a população B utiliza um AE com as funções da primeira coluna. O valor entre parênteses representa o erro amostral.

Tabela 2: Comparação dos diferentes AEs

AE da população B	txdom(A)	Empate	txdom(B)
Hereditariedade, Predação Randômica e Síntese	0,42749 (0,02094)	0,10863 (0,01792)	0,46387 (0.02016)
Predação Síntese	0,41276 (0,02031)	0,1169 (0,01760)	0,47033 (0,02232)
Predação Randômica	0,42651 (0,02126)	0,09234 (0,01936)	0,48115 (0,02096)
Predação Randômica e Síntese	0,43253 (0,01917)	0,07385 (0,01203)	0,49362 (0,01948)
Hereditariedade	0,38007 (0,01727)	0,10060 (0,01699)	0,51934 (0,02021)

A Tabela 2 possibilita identificar que todos os AEs implementados se mostraram mais aptos a adaptar as populações do que um AE tradicional (contendo apenas crossover e mutação). Seguindo os resultados encontrados, o AE que conseguiu dominar com mais facilidade a população controlada pelo AE tradicional foi aquele contendo apenas a função evolutiva "Hereditariedade", pois apresentou os melhores resultados em simulação (Taxa de dominação de aproximadamente 52% com 2% de erro amostral). A vantagem desse algoritmo é que o mesmo é mais resistente à queda de pontuação da população evoluída. No entanto, a função hereditariedade faz com que uma boa solução encontrada por um indivíduo demore mais para ser difundida na população. A adaptação apresentada nesse algoritmo foi mais lenta e mais difícil de notar do que nos outros algoritmos utilizados.

O algoritmo que apresentou uma adaptação mais fácil de ser percebida e que, portanto, foi o escolhido para ser o algoritmo utilizado no jogo, foi o que utiliza as técnicas de Predação por Síntese e Randômica. Uma curiosidade que ocorreu ao se utilizar esta técnica foi que as criaturas verdes (herbívoras) começaram a "predar" a criatura do jogador. Tal fato ocorreu porque ouve um erro de projeto na primeira versão do jogo, em que o autor esqueceu de reprogramar a função Predação Randômica para que não modificasse o atributo "Herbcarn" das criaturas. Tal erro já foi reparado, mas, no entanto, ilustra bem a capacidade do AE de explorar as oportunidades oferecidas (incluindo erros no código) para gerar melhores soluções, tentando vencer a qualquer custo o jogador.

O objetivo do trabalho era criar maneiras pelas quais as criaturas controladas pelo computador desenvolvessem estratégias de jogo que se tornassem desafiadoras e interessantes ao jogador. Alguns

comportamentos emergentes desenvolvidos pelas criaturas observadas ao se executar o jogo por várias gerações são descritos a seguir:

• Predadores “guardando” as árvores (jogando no Modo2): além do comportamento de perseguição esperado, os predadores, em algumas ocasiões do jogo, utilizaram a estratégia de esperar ao lado das plantas. Quando o jogador ou uma criatura herbívora controlada pelo computador se aproximava da planta “guardada” pelo carnívoro, este se aproveitava da situação para se alimentar do “invasor”, protegendo a planta que continuava servindo como isca para atrair outras criaturas. Esse é um comportamento classificado na biologia como Mutualismo, no qual a planta e o indivíduo estão associados para o bem comum. A Figura 4 mostra esse comportamento.

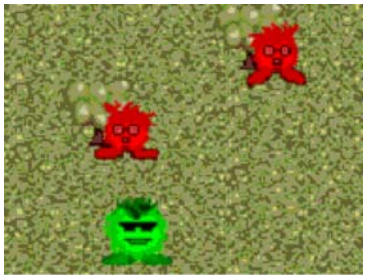


Figura 4: Criaturas carnívoras (vermelhas) guardando as árvores que atraem as criaturas herbívoras (verde).

• Formação de “enxames”: as criaturas herbívoras, além dos comportamentos esperados de evasão do personagem do jogador e perseguição de plantas, apresentaram em algumas ocasiões comportamentos de agrupamento em enxames, resultante da perseguição de membros da população pelos carnívoros. Dessa forma, mesmo quando a criatura não possuía a habilidade de se evadir do personagem do jogador, se uma criatura do grupo possuísse tal habilidade, as outras poderiam segui-la, apresentando um desafio ao jogador. Outra ocorrência resultante desse comportamento eventualmente ocorria quando o jogador perseguia o grupo e o mesmo se dividia para fugir. Esse comportamento pode ser visualizado na Figura 5.

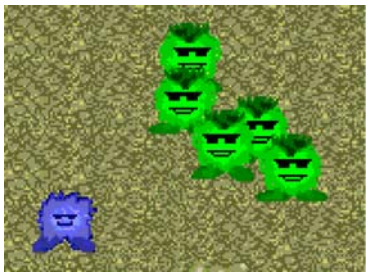


Figura 5: Agrupamentos formados pelas criaturas herbívoras

• Adaptação dos comportamentos das criaturas ao jogador (jogando no Modo1): A Figura 6 representa um dos ciclos observados que ilustra a capacidade de adaptação on-line do algoritmo. As Elipses são os comportamentos adotados pelas criaturas herbívoras evoluídas pelo AE e os retângulos indicam o comportamento adotado pelo jogador. O jogador, no início do jogo, pode priorizar a perseguição de plantas ou a perseguição de criaturas. Caso persiga criaturas, as mesmas passam a priorizar fugir do jogador a perseguir plantas. Ou seja, quando tanto o jogador quanto uma planta se encontrarem no raio de ação da criatura, a atitude tomada pela mesma é fugir do jogador. Dessa forma, o jogador desiste de perseguir as criaturas, já que não mais as alcança, e passa a perseguir as plantas para sobreviver. Assim, as criaturas percebem que podem priorizar perseguir plantas, passando a comê-las antes do jogador e, conseqüentemente, matando-o de fome. Nesse momento, o jogador se vê obrigado mais uma vez a perseguir as criaturas antes que as mesmas comam as plantas, fechando o ciclo.

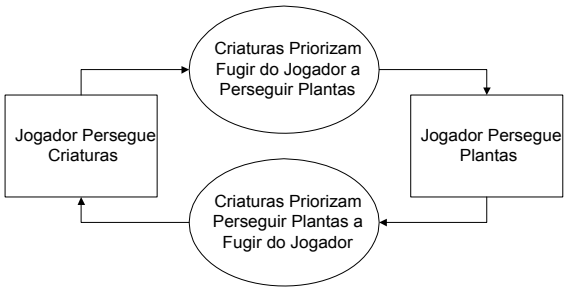


Figura 6: Adaptação dos comportamentos adotados pelas criaturas em função das estratégias adotadas pelo jogador.

9. Conclusão

O Algoritmo de aprendizado desenvolvido durante este trabalho foi capaz de satisfazer o critério de eficiência estabelecido por Spronck para um aprendizado on-line que funcione na prática, já que durante os experimentos com jogadores foi possível notar a adaptação do computador em tempo de jogo. Este trabalho apresenta, portanto, um contra-exemplo para a afirmação de que AEs não podem satisfazer o critério de eficiência no aprendizado on-line. No entanto, o algoritmo desenvolvido não satisfez o critério de efetividade apontado por Spronck, pois, durante a evolução, os personagens do computador podem adotar estratégias consideradas “inadequadas” pelo jogador. Um exemplo disso acontece quando presas passam a perseguir seus predadores. Tal comportamento se deve ao fato da adaptação ocorrer de forma direta, atuando nos comportamentos mais básicos presentes nos agentes do computador. Enquanto esse tipo de adaptação permite que estratégias “inadequadas” possam ser geradas pelo computador, ela também permite o surgimento de novas estratégias, muitas vezes não previstas pelo programador (como por exemplo aquelas apresentadas nas Figuras 4 e 5). Dessa forma, não é necessário que o programador seja um especialista no jogo, tendo que codificar todas as estratégias que serão utilizadas e é de se esperar que a capacidade de gerar estratégias novas a cada jogo diferente renove o interesse despertado pelo mesmo em seus usuários.

O resultado final deste trabalho culminou também no início de um novo projeto, no qual um novo jogo será construído, assim como um novo AE responsável por seu aprendizado, que terá como meta satisfazer também o requisito de efetividade apontado por Spronck. Em suma, o objetivo desse novo projeto será o de responder a pergunta: é possível construir um AE para o aprendizado on-line de jogos que possa ser utilizado na prática, por satisfazer os requisitos de rapidez, efetividade, robustez e eficiência?

Agradecimentos

O presente trabalho foi realizado com apoio do CNPq, Conselho Nacional de Desenvolvimento Científico e Tecnológico – Brasil.

Os autores gostariam de agradecer a Francisco Krug pela música produzida para o jogo.

Referências

BRAMLETTE, M., 1991. Initialization, mutation and selection methods in genetic algorithms for function optimization. *In: Proceedings of the Fourth International Conference on Genetic Algorithms*. San Mateo, CA, USA, 100–107.

CROCOMO, M., SIMÕES, E.V. AND DELBEM, A.C.B., 2004. Otimização automática de parâmetros de sistemas evolucionários de acordo com a aplicação. *In: Anais do I Encontro de Robótica Inteligente do XXIV Congresso da Sociedade Brasileira de Computação.* Salvador, BA.

DARWIN, C., 1998. *The origin of species by means of natural selection*. Reading, UK: Cox & Wyman (reprint from the sixth edition), 458 p., ISBN 185958070.

ELKAN, C., 2001. Paradoxes of fuzzy logic, revisited. *In: International Journal of Approximate Reasoning,* 26(2), 153–155.

FRANCISCO, W., 1993. *Estatística: síntese de teoria, exercícios propostos e resolvidos*. Piracicaba:Unimep, 219 p.

HAYKIN, S., 2001. *Redes neurais: princípios e prática*. Bookman, 900 p., ISBN 8573077182.

HOLLAND, J., 1962. Outline for a logical theory of adaptive systems. *In: Journal of the Association for Computing Machinery*, 3, 297–314.

KOZA, J., 1992. Genetic evolution and co-evolution of game strategies. *In: International Conference on Game Theory and Its Applications*, New York, USA.

LUCAS, S., 2004. Cellz: a simple dynamic game for testing evolutionary algorithms. *In: Proceedings of the 2004 IEEE Congress on Evolutionary Computation*, IEEE Press, 1007–1014.

MANSLOW, J., 2002. Learning and adaptation (book chapter). *AI game programming wisdom*, Charles River Media, 557–566.

MITCHELL, M. AND FORREST., S., 1994. Genetic algorithms and artificial life. *In: Artificial Life*, 1(3), 267–289.

PICCIONE, M. AND RUBINSTEIN, A. 1993. Finite automata play a repeated extensive game. *In: Journal of Economic Theory*, 61, 160–168.

PONSEM, M., 2004. *Improving adaptive game AI with evolutionary learning*. MSc thesis, Faculty of Media & Knowledge Engineering, Delft University of Technology, Netherlands.

PRECHELT, L., 2003. Are scripting languages any good? A validation of Perl, Python, Rexx, and Tcl against C, C++, and Java. *In: Advances in Computers*, 57, 207–271, ISSN 0065-2458.

REYNOLDS, W. C., 1987. Flocks, Herds, and Schools: A Distributed Behavioral Model. *In: Proceedings of SIGGRAPH*. Anaheim, CA, USA, 25–34.

SIMÕES, E.V., 2000. *Development of an embedded evolutionary controller to enable collision-free navigation of a population of autonomous mobile robots*. PhD thesis, University of Kent at Canterbury, UK.

SIMÕES, E.V. AND BARONE, D.A.C., 2003. Predation: an approach to improving the evolution of real robots with a distributed evolutionary controller. *In: Proceedings of International Conference on Robot Automation.* New York: IEEE Press, 664–669.

SIMÕES, E.V. AND DIMOND, K., 1999. An evolutionary controller for autonomous multi-robot systems. *In: Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, Tokyo, Japan, 596–601.

SPRONCK, P., YEE, N., SHPRINKHUIZEN-KUYPER, I. AND POSTMA, E., 2004. Online adaptation of game opponent AI in theory and practice. *In: Proceedings of the 4th International Conference on Intelligent Games and Simulation*, London, UK, 45–53.

SUTTON, R. AND BARTO, A., 1998. *Reinforcement learning: an introduction*. MIT Press, 432 p.

SWEETSER P., 2002. *Current AI in games: a review*. Technical report, School of ITEE, University of Queensland, Australia.

TOMASSINI, M., 1995. Survey of genetic algorithms. *In: Annual Reviews of Computational Physics*, World Scientific, 3, 87–118.

WATSON, R., FICICI, S. AND POLLACK, J., 2002. Embodied evolution: distributing an evolutionary algorithm in a population of robots. *In: Robotics and Autonomous Systems*, 39, 1–18.

YANNAKAKIS, G.N., 2005. *AI in computer games: generating interesting interactive opponents by the use of evolutionary computation*. PhD thesis, University of Edinburgh, UK.

YAO, X. AND DARWEN, P., 2000. Genetic algorithms and evolutionary games. *In: Commerce, Complexity and Evolution*, Cambridge: University Press, 1(16), 313-333.

Implementação de Suporte a Modelos de Personagem Não Jogador em Dispositivos Móveis na Mobile 3D Game Engine

Paulo César Rodacki Gomes
Cláudio José Estácio
FURB - Universidade Regional de Blumenau
DSC - Departamento de Sistemas e Computação

Vitor Fernando Pamplona
UFRGS - Universidade Federal do Rio Grande do Sul
PPGC - Programa de Pós-Graduação em Computação
Instituto de Informática

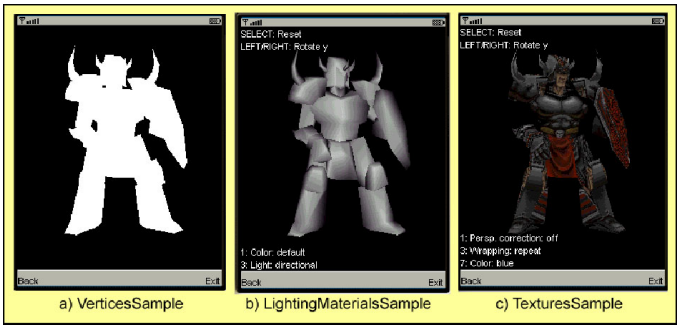


Figura 1: *Personagem Knight em 3 diferentes modos de visualização*

Resumo

Nos últimos anos, jogos 3D para celulares têm recebido significativa atenção dos pesquisadores e incentivos fiscais do estado brasileiro. Ainda longe da qualidade gráfica dos jogos para computadores e consoles, os jogos para celulares, que sofrem pela severa limitação de memória e processamento dos aparelhos modernos, caminham a passos lentos. Recentes investidas contra estas difuldades criaram motores de jogos [Gomes and Pamplona 2005], [de Albuquerque Macêdo Jr. 2005] com a finalidade de facilitar o desenvolvimento e otimizar os jogos. Este trabalho apresenta a implementação do módulo de importação e visualização de Personagens Não Jogador (PNJs) em um dos motores de jogos citados anteriormente, o *Mobile 3D Game Engine* (M3GE). Desta forma, foi adicionado o suporte a um formato de modelos chamado *Quake II's Models* (MD2), muito utilizado para a modelagem de personagens para jogos, pois incorpora recursos de animação quadro a quadro. O trabalho não discute questões relativas à inteligência dos PNJs. São apresentados detalhes da implementação de importação e visualização dos modelos, bem como a demonstração de exemplos e testes de desempenho de forma a validar a solução proposta.

Keywords:: Personagem não Jogador (PNJ), Motores de Jogos, M3G, Dispositivos Móveis, MD2

Author's Contact:

rodacki@inf.furb.br
{claudioestacio, vitorpamplona}@gmail.com

1 Introdução

Os jogos eletrônicos para dispositivos móveis são uma opção de entretenimento que vem despertando cada vez mais interesse tanto por parte dos jogadores quanto pela indústria de software brasileira. Tal categoria de jogos vem se modernizando constantemente, seguindo a evolução do hardware, notoriamente dos telefones celulares, onde dentre os recursos mais importantes, estão as recentes implementações e melhorias nas rotinas de animação 3D.

Sob uma ótica retrospectiva, em poucos anos os jogos para consoles e computadores evoluíram na simplicidade dos gráficos em duas dimensões (2D) e sons reproduzidos com algumas notas de bips, para um mundo mais realista, com visualização de ambientes 3D, com mais ação e sons com recursos interativos e 3D. Visto esse processo evolutivo dos jogos para computadores, pode-se esperar que o mesmo deva acontecer nos jogos para celulares. Atualmente a maioria dos jogos nesta plataforma são em 2D, mas já existem diversas iniciativas para desenvolvimento em 3D. Para facilitar o desenvolvimento dos jogos, utilizam-se os motores, que são bibliotecas de software responsáveis pelos ciclos de interação existentes em um jogo, as quais são capazes de tratar a maioria das necessidades existentes na sua implementação. A Mobile 3D Game Engine (M3GE) [Gomes and Pamplona 2005] é um motor de jogos para telefones celulares baseado na Java Mobile 3D Graphics API (M3G) [Nokia 2004] e já possui diversas funcionalidades implementadas, tais como, importação e renderização de ambientes 3D, criação de câmeras, tratamento de eventos, movimentação de personagens e câmeras pelo cenario e tratamento de colisão. Cabe ressaltar que a M3GE é a primeira implementação livre de motor de jogos 3D em Java que se tem notícia.

À medida que os jogos em dispositivos móveis vão se tornando mais complexos e sofisticados, o nível de realismo gráfico deverá levar os jogadores a acreditar que os jogos ocorrem em mundos muito mais realistas. Um componente importante desse crescente nível de realismo são os Personagens Não Jogador (PNJs). Os PNJs são os personagens presentes no jogo e que não são controlados pelo jogador, mas que se relacionam de alguma forma com ele. Por exemplo, num jogo de primeira pessoa, os PNJs são os personagens inimigos do jogador, que devem ser acertados por seus disparos.

Este artigo apresenta detalhes da implementação de uma extensão da M3GE que consiste na integração de Personagem Não Jogador (PNJ). Tal integração é feita com recursos de importação de modelos sofisticados, incorporação dos mesmos ao grafo de cena disponibilizado pela M3G, e, naturalmente, a renderização destes modelos na cena. A implementação considera a importação de modelos de personagens animados no formato de arquivo *Quake II's Models* (MD2) [Henry 2004]. O MD2 é um dos formatos mais populares para inclusão de PNJs em jogos 3D. O presente trabalho não aborda questões de suporte a inteligência artificial para os PNJs.

O artigo está estruturado da seguinte forma: primeiramente, a seção de introdução contextualiza o problema, e as duas próximas seções apresentam uma breve revisão das principais tecnologias para desenvolvimento de jogos para telefones celulares, incluindo informações gerais a respeito de motores de jogos. A seguir, são comentados alguns trabalhos correlatos, detalhando-se as principais características do motor de jogos *Mobile 3D Game Engine* (M3GE). Após, é apresentado o conceito de Personagem Não Jogador e detalhes dos modelos MD2. Finalmente, é discutida a implementação de suporte à importação de modelos MD2 na M3GE, sendo posteriormente apresentados os resultados e conclusões do artigo.

2 Jogos em Dispositivos Móveis

O avanço da tecnologia dos celulares nos últimos anos fez com que esses aparelhos deixassem de ser apenas usados para transmissão e recepção de voz e passassem a ter diversas utilidades. O desenvolvimento de jogos para tais dispositivos vem ganhando cada vez mais importância, e as soluções mais utilizadas pela comunidade de desenvolvedores são a Sun J2ME, a Qualcomm BREW e a Mophun [Paiva 2003].

A Sun Java 2 Micro Edition (J2ME) é uma versão da linguagem Java para dispositivos móveis, sendo a primeira iniciativa nesse segmento em 2000. Para sua aplicação rodar em um celular, basta que o celular possua uma máquina virtual Java, como a Kilobyte Virtual Machine (KVM) [Müller et al. 2005].

Em janeiro de 2001 a Qualcomm lançou o Binary Runtime Enviroment for Wireless (BREW). Para este ambiente as aplicações são desenvolvidas em C/C++, e depois de compiladas rodam em um chip separado do processador principal do dispositivo. Esta arquitetura apresenta características semelhantes a J2ME por apresentar suporte a APIs OpenGL ES [Khronos Group 2004]. Ainda existe a Application Program Interface (API) desenvolvida pela própria empresa, a QX Engine, que tem recursos que facilitam o desenvolvimento de jogos. Uma característica interessante apresentada pelo BREW é que para uma aplicação ser comercializável, ela deve estar cadastrada na empresa e passar por rigorosos testes, para evitar que haja erros em tempo de execução. Isto viabiliza uma certificação de qualidade e originalidade à aplicação [Müller et al. 2005].

O padrão Mophun foi desenvolvido pela sueca Synergenix. É menos usado em relação ao Java e ao BREW, mas tem como característica deixar os jogos menores, os quais ficam entre metade e um terço do tamanho das outras duas soluções [Paiva 2003].

3 Motores de Jogos 3D

Um motor de jogo é o coração de qualquer jogo eletrônico. Inicialmente os motores eram desenvolvidos para fazer parte do jogo e serem usados somente para este fim específico. Visto que diferentes jogos têm similaridades em seu processo de desenvolvimento, os motores passaram a implementar funcionalidades comuns a determinados tipos de jogos. Assim seu código fonte é reutilizado, isto é, funções que foram usadas num determinado jogo, podem ser usadas em outros que apresentam características semelhantes [Harrison 2003].

Conforme definido por Macêdo Júnior [de Albuquerque Macêdo Jr. 2005], um motor de jogos 3D é um sistema que produz uma visualização em tempo real respondendo aos eventos da aplicação. Para o funcionamento desse processo deve-se utilizar várias técnicas, algoritmos, estruturas de dados e conhecimentos matemáticos. Estes aspectos ficam transparentes quando se desenvolve um jogo utilizando motor de jogos 3D pois o desenvolvedor fica abstraído dessas funcionalidades, preocupando-se apenas com a lógica e o enredo do jogo.

O motor disponibiliza as principais características para o desenvolvimento de jogos, como por exemplo: renderização de cenários 3D, recursos de comunicação em rede, tratamento de eventos, mapeamento de texturas, *scripting*, inteligência artificial, entre outros [Finney 2004]. O principal foco dos motores 3D está na qualidade e no desempenho computacional da apresentação visual (*rendering*) do jogo. Como os ambientes são tridimensionais, diversos cálculos são usados para renderizar cada quadro da visualização, juntando-se os vértices com as texturas e aplicando efeitos. Alguns exemplos de motores de jogos 3D são: Crystal Space [Crystal Space 2004], Ogre 3D [Ogre3D 2005], Genesis3D [Eclipse Entertainment 2004], 3D Game Studio [Conitec Corporation 2004] e Fly3D [Paralelo Computação 2004].

A figura 2 apresenta a arquitetura de um motor de jogos 3D. O gerenciador principal é o centro dos demais níveis, sendo o responsável por interligar cada funcionalidade do motor de jogos. O gerenciador de entrada organiza e repassa as informações vindas do usuário. O gerenciador gráfico apresenta a cena atual na saída do dispositivo, neste caso a tela. O gerenciador de inteligência artificial gerencia os personagens não jogadores, definindo seus movimentos. O gerenciador de múltiplos jogadores gerencia a comunicação com os demais jogadores. O gerenciador de mundo armazena o grafo de cena do jogo onde estão interligados todos os objetos. O gerenciador de objetos tem a finalidade de inserir, alterar e excluir os objetos do jogo. O objeto do jogo armazena as informações de cada entidade do jogo. O gerenciador de som controla os efeitos sonoros provenientes dos objetos, enviando para a saída, que neste caso são os alto-falantes do dispositivo. E por último o editor de cenário é um nível a parte do restante que tem por finalidade criar a estrutura inicial da visualização dos jogos, composta pelos chamados "mapas". Também pode ser realizada no editor de cenário a criação dos personagens 3D [Gomes and Pamplona 2005].

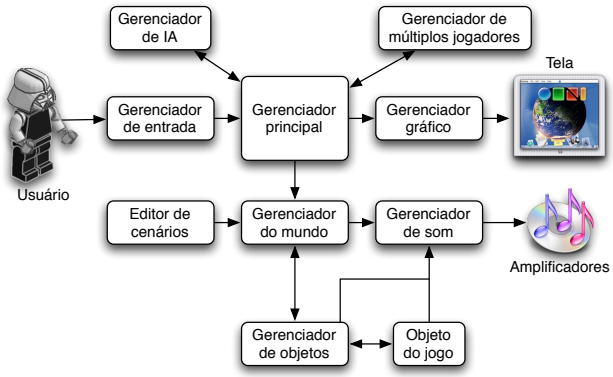


Figura 2: *Arquitetura de um motor de jogos*

4 Trabalhos Correlatos

O desenvolvimento de motores de jogos para dispositivos móveis é uma área ainda recente, mas podemos citar três projetos acadêmicos relacionados ao presente trabalho: o wGEN, o mOGE e a M3GE.

O wGEN é um *framework* de desenvolvimento de jogos 2D para dispositivos móveis [Pessoa 2001]. Ele é o pioneiro na utilização de J2ME num motor de jogos, porém não tem recursos 3D. O wGEN apresenta como componentes e funcionalidades: representação do objeto do jogo, representação do mapa do jogo, gerenciamento de objetos, gerenciamento de entrada, gerenciamento gráfico, gerenciamento do jogo, gerenciamento de rede, gerenciamento de aplicação e integração com o editor de cenários.

Outro motor de jogos com características semelhantes com este trabalho é o Mobile Graphics Engine (mOGE) [de Albuquerque Macêdo Jr. 2005]. Este projeto teve como objetivo prover diversas técnicas de computação gráfica 3D. Para a renderização, o mOGE utiliza o *pipeline* gráfico, que é a decomposição do processo de transformação de objetos 3D em imagens, composto pelos estágios de aplicação, de geometria e de rasterização. Para a representação de objetos foi utilizada malha de triângulos. Para a estrutura dos objetos presentes no jogo é utilizado um grafo de cena. Da mesma forma que na M3GE, a utilização de um grafo de cena facilitou o processo de animação dos objetos. Este motor de jogos apresenta ainda projeção de câmeras, detecção de colisão e efeitos especiais baseados em imagens [de Albuquerque Macêdo Jr. 2005].

No âmbito de jogos comerciais, o motor 3D mais conhecido é o Edgelib [Elements Interactive 2007]. Desenvolvido em C++ para sistema operacional Symbian pela empresa Elements Interactive, este motor já é bastante usado por uma grande quantidade de desenvolvedores. Possui recursos para gráficos 2D e 3D, com suporte para OpenGL ES, além de recursos para desenvolvimento de jogos em rede, áudio, gerenciamento de memória e de arquivos, entre outros.

Recentemente foi anunciado pela empresa Fishlabs o motor de jogos Abyss 2.0 [FISHLABS Entertainment GmbH 2007]. Compatível com celulares que rodem a JSR-184 [Nokia 2003], este motor apresenta recursos de renderização 3D, juntamente com um motor de Inteligência Artificial e Física de corpos Rígidos.

Nenhum dos motores citados possuem funcionalidades específicas

para a importação de PNJs, porém o grafo de cena do mOGE é um dos recursos úteis para uma eventual implementação de inserção de PNJs nos jogos, de forma que o motor possa desenhá-los na cena.

5 Mobile 3D Game Engine

A Mobile 3D Game Engine (M3GE) é um protótipo de motor de jogos para dispositivos móveis com suporte a M3G. Ela apresenta diversas funcionalidades implementadas, tais como: importação e renderização de ambientes 3D, criação de câmeras, tratamento de eventos, movimentação de personagens no cenário e tratamento de colisão. Na página do projeto (`https://m3ge.dev.java.net/`), é disponibilizado um exemplo de um jogo com as funcionalidades implementadas, onde um cenário 3D com objetos estáticos pode ser visualizado. A figura 3 demonstra duas telas desta aplicação exemplo. O jogador, representado por um cubo, é controlado pelos comandos do teclado do celular. A movimentação do jogador está sujeita ao tratamento de colisões e possui capacidade de disparar tiros nos demais objetos da cena. A implementação da M3GE é feita sobre a M3G que por sua vez é implementada através da J2ME. A M3G é uma API Gráfica 3D para ser utilizada em dispositivos móveis com suporte a programação Java. Esta foi criada em função da maioria dos jogos para esse tipo de dispositivo ser em 2D e haver uma demanda latente por jogos 3D em celulares [Höfele 2005].

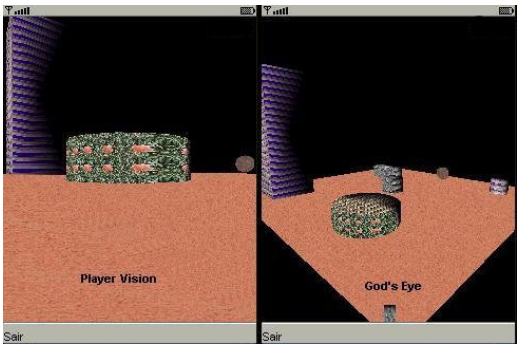


Figura 3: *Aplicação exemplo da M3GE*

O motor M3GE foi projetado anexo à M3G, significando que se pode acessar diretamente a M3G na aplicação, se necessário. Isto dá maior flexibilidade na implementação dos jogos. O motor foi dividido em dois grandes componentes: o responsável pela leitura dos arquivos de entrada e o *core*, além de um terceiro componente de classes utilitárias. A figura 4 ilustra a arquitetura da M3GE, seus principais componentes e classes. Para se montar o cenário do jogo são lidos três tipos de arquivos: um com as configurações gerais do ambiente, um com as malhas de polígonos e outro com uma série de texturas. As classes ObjLoader e MbjLoader são responsáveis por gerenciar os objetos, elas criam os objetos da classe Object3D e os nós do grafo de cena. A classe Object3D é a classe base de todos os objetos 3D visíveis da M3G. Ela inclui o nodo pai ou qualquer outro nodo no grafo de cena, uma animação, uma textura, entre outros. Em cada nodo do grafo estão presentes um ou mais grupos, que por sua vez são interligados aos objetos 3D da classe Object3D. Estes objetos podem ser modelos gráficos 3D, câmeras, luzes, imagens 2D, entre outros.

O componente *core*, na figura4 é o motor de jogos propriamente dito, que tem como gerenciador de entrada a classe KeysManager. A classe Player implementa o personagem principal do jogo. A classe EngineCanvas é o gerenciador principal. Para criar as câmeras existe a classe Cameras. Para fazer o tratamento de colisão trabalham juntas as classes Configuration e CollisionDetection. A classe UpdateListener deve ser implementada pelo desenvolvedor do jogo para tratar funcionalidades na renderização, ou da lógica através da adição de eventos. E a classe UpdatesceneTimertask implementa rotinas para a chamada de renderização num determinado intervalo de tempo.

No pacote de carga de arquivos, a classe Loader é a interface para duas classes, a ObjLoader e a MbjLoader. A M3G utiliza arquivos no formato Wavefront (obj) para importação de modelos 3D, que são carregados pela classe ObjLoader. Experiências anteriores constataram que o processo de importação de modelos em telefones celulares é demasiadamente lento, pois cada arquivo (de geometria e de materiais) precisa ser lido até três vezes. Por isso, a M3GE utiliza um formato próprio, o Mobile Object File (mbj), no qual uma série de informações sobre o modelo já estão pré-processadas, agilizando a sua importação. A classe MtlLoader é utilizada para a carga deste tipo de arquivos, que descrevem cores, propriedades de materiais e texturas do ambiente ou de grupos de objetos 3D. Os arquivos mtl podem ser utilizados juntamente com arquivos Wavefront, permitindo que modelos em ambos os formatos sejam carregados em um mesmo jogo. A classe FileReader é uma classe especial para facilitar a leitura dos arquivos.

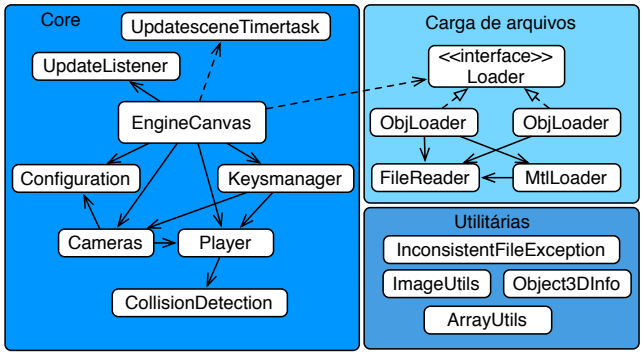


Figura 4: *Arquitetura da M3GE*

A M3GE ainda conta com um conjunto de classes utilitárias. A classe InconsistentFileException lança uma exceção quando ocorre algum erro na leitura de arquivos. A classe ImageUtils é utilizada para fazer a leitura de um arquivo de imagem utilizado como textura. A classe Object3DInfo é utilizada para armazenar o ponto central de um objeto 3D e seu respectivo nome. Por último, a classe ArrayUtils é utilizada para chamar métodos para tratamento de listas.

As conclusões relatadas por Gomes e Pamplona [Gomes and Pamplona 2005] afirmaram que Java é promissora para a implementação de jogos para dispositivos móveis. Apesar da M3GE ser apenas um protótipo, ela já vem sendo utilizada para desenvolvimento de jogos comercialmente [Guiliano 2005]. Além disso, recentemente foi implementado um módulo para jogos multiusuário com comunicação via *bluetooth* [Carandina and Martins 2006].

6 Personagem Não Jogador

O termo Personagem Não Jogador (PNJ) vem do inglês *Non-Player Character* (NPC), e define-se por um personagem presente no jogo, que não é controlado pelo jogador, mas que se relaciona de alguma forma com ele. Os PNJs podem ser elementos decorativos das cenas ou interagir diretamente com as ações do jogador [Sánchez and Dalmau 2004]. Para jogos onde o usuário joga contra o computador, usam-se os PNJs para representar oponentes. Por exemplo, num jogo de primeira pessoa existem na cena personagens que são inimigos do jogador e tem-se como objetivo acertar tiros neles. Estes personagens podem ter diversas aparências, isto depende dos modelos que foram desenhados e também de suas texturas. Deste modo, tem-se no jogo a possibilidade de interação com diversos inimigos, alguns mais parecidos com humanos, outros com figuras mitológicas, mutantes, robôs, entre outros. Estes personagens são criados com relação ao enredo do jogo.

O primeiro jogo a incluir um oponente animado por computador foi *Shark Jaws* da empresa Atari em 1974. Nesse jogo, o PNJ é um tubarão e tem como ação perseguir o mergulhador controlado pelo jogador. Para isso, foi usada Inteligência Artificial (IA) de uma forma simplificada, se comparada com o que existe nos jogos atuais [Byl 2004].

7 Modelo Animado MD2

O formato de arquivo de modelo MD2 é um dos formatos mais populares para inclusão de PNJs em jogos 3D [Henry 2004]. Surgiu em novembro de 1997 para ser utilizado no jogo *Quake II* produzido pela ID Software. Suas principais características são: ser um modelo geométrico formado por triângulos, utilizar animação quadro a quadro, isto é, cada quadro representa uma pose do personagem, e quando as poses são exibidas em seqüência dá-se a visualização do movimento. Um outra característica importante é que sua estrutura de dados já foi concebida contendo primitivas gráficas do OpenGL para desenho do modelo, onde os vértices já estão organizados para facilitar a chamada de desenho de primitivas do tipo Gl_TRIANGLE_FAN e Gl_TRIANGLE_STRIP. Estas primitivas têm utilização opcional, pois além da lista de vértices, o arquivo possui uma lista dos triângulos utilizados na modelagem do personagem.

Este formato é uma variação dos arquivos MDL que são usados em jogos como *Half Life* e no *Counter Strike*, que utilizam uma versão derivada do motor *Quake II* [Santee 2005]. A extensão do arquivo do modelo é MD2 e ele está em formato binário dividido em duas partes: cabeçalho e dados, comentadas a seguir.

7.1 Cabeçalho

Conforme descrito por Santee [Santee 2005], é no cabeçalho do arquivo MD2 que estão definidos os números de vértices, faces, quadros chave e tudo mais que está dentro do arquivo. Pelo fato de ser um arquivo binário, o cabeçalho torna-se muito importante para a localização do restante dos dados. Cada informação presente no cabeçalho é armazenada em quatro *bytes* representados por um tipo de dado inteiro.

Antes de apresentar o conteúdo do cabeçalho é preciso rever alguns conceitos:

- Quadro-chave: a animação do modelo é representada por uma seqüência de quadros-chave, e cada um representando uma posição do personagem, devendo ser realizada interpolação entre os quadros-have. Para armazenar um quadro-chave, o modelo guarda uma lista de vértices, o nome do quadro, um vetor de escala e um vetor de translação. Este dois últimos são usados para a descompactação dos vértices, multiplicando cada coordenada do vértice pelo vetor de escala e somando ao vetor de translação. Isto é feito para transformar os vértices de ponto fixo para ponto flutuante;
- Coordenadas de Textura: as coordenadas de textura são utilizadas para mapear a textura no modelo 3D. Isto é feito através das coordenadas s e t. Cada vértice do modelo 3D recebe um par de coordenadas de textura s e t;
- *Skin*: são texturas que dão representação visual externa ao modelo. Através dos *skins* podem-se distinguir partes do corpo, roupa, tecido, materiais anexados, entre outros. Esses *skins* são mapeados para o modelo através das coordenadas de textura. No cabeçalho encontram-se o tamanho e a quantidade de *skins* que podem ser utilizados no modelo;
- Comando OpenGl: os comandos OpenGl presentes no arquivo são usados para a renderização através das primitivas GL_TRIANGLE_FAN e GL_TRIANGLE_STRIP. O arquivo MD2 apresenta uma lista de números inteiros, que estão na seguinte estrutura: o primeiro valor é o número de vértices a ser renderizado com a primitiva GL_TRIANGLE_FAN caso seja positivo ou renderizado com a primitiva GL_TRIANGLE_STRIP caso seja negativo. Os seguintes números vêm na seqüência divididos em grupos de três, onde os dois primeiros representam as coordenadas de textura s e t, e o terceiro o índice para o vértice. Repete-se a estrutura acima até que o número de vértices a desenhar seja zero significando o fim da renderização do modelo;
- Triângulos: são os que definem as faces do modelo. Eles são uma lista de índices da lista de vértices. Cada conjunto de três índices representa um triângulo;
- Vértices: são os pontos no mundo cartesiano que desenham o modelo 3D. São representados pelas coordenadas: x, y e z;
- Vetor Normal: é um vetor que está contido na estrutura dos vértices, utilizado para calcular a iluminação do modelo.

A estrutura completa do cabeçalho pode ser vista na tabela 1. Esta estrutura representa a mesma seqüência presente dentro do arquivo MD2.

Tabela 1: *Cabeçalho dos arquivos MD2*

Conteúdo	Descrição
magic	ID do formato (deve ser igual a 844121161)
version	Versão do arquivo (para Quake2 precisa ser igual a 8)
skinWidth	Largura do skin do modelo (*pixels*)
skinHeight	Altura do skin (em *pixels*)
frameSize	Tamanho das informações dos frames (em *bytes*)
numSkins	Número de skins avaliados
numVertices	Número de vértices
numTexCoords	Número de coord. de textura
numTriangles	Número de triângulos
numGlCommands	Número de comandos OpenGL
numFrames	Número de quadros para animação
offsetSkins	Posição dos skins
offsetTexCoords	Posição das coordenadas de textura no *buffer*
offsetTriangles	Posição dos triângulos no *buffer*
offsetFrames	Posição dos quadros
offsetGlCommands	Posição dos comandos OpenGL
offsetEnd	Posição para o final do modelo

7.2 Dados

As informações de cada vértice do modelo são armazenadas em poucos *bytes*. O resultado disso é um arquivo pequeno, mas com perdas na exatidão da geometria do modelo [MD2 2005]. Dispositivos móveis têm espaço de memória restrita e a resolução dos *displays* nesses dispositivos é bem menor do que em um monitor de um computador do tipo PC ou de uma tela de aparelho de TV. Por isso a perda de qualidade na visualização dos modelos não é tão perceptível quanto seria em um computador *desktop*. Os tipos de dados são apresentados da seguinte forma:

- Vetor: composto por três coordenadas do tipo *float*;
- Informação Textura: lista de nomes de texturas associadas ao modelo;
- Coordenadas da Textura: são armazenadas na estrutura como *short integers*;
- Triângulos: cada triângulo possui uma lista com os índices dos vértices e uma lista com os índices das coordenadas da textura;
- Vértices: são compostos de coordenadas 3D, onde são armazenados em um *byte* cada coordenada e um índice do vetor normal;
- Quadros: têm informações específicas da lista de vértice do quadro;
- Comandos OpenGL: são armazenados em uma lista de *integers*.

7.3 Texturas

As texturas utilizadas nos modelos MD2 são localizadas em arquivos separados. Todos os vértices do modelo devem ser mapeados para essa mesma textura. Por este motivo a textura deve ter diferentes regiões para representar cada parte do personagem, desde os pés até a cabeça, chamados assim de *skins* [Misfit Code 2005]. Na

figura 5 pode-se visualizar um arquivo de textura e ao lado o modelo utilizando-se dessa textura. O mesmo personagem pode utilizar diversos *skins* diferentes para criar novas visualizações. Pode-se comparar a troca de *skin* do personagem à troca de roupa de uma pessoa.

Figura 5: *Exemplo de skin*

O nome da textura é armazenado em 64 *bytes* do tipo *string*. Os *skins* são normalmente usados no formato de arquivo PCX, porém pode-se encontrar em outros diversos formatos, tais como PNG, BMP, GIF, entre outros.

8 Implementação de PNJs na M3GE

A especificação do módulo para PNJs foi feita com utilização dos diagramas de caso de uso, de atividades, de classes e de seqüência da UML. A figura 6 ilustra os dois casos de uso implementados nesta primeira versão, representando as funcionalidades que o jogo pode realizar a partir da M3GE utilizando a importação de modelos MD2. O ator representado pelo jogo possui dois casos de uso. O primeiro tem como função fazer a leitura e importação do arquivo MD2. Após os dados serem lidos, o segundo caso de uso demonstra que o personagem 3D deve ser exibido na tela do jogo.

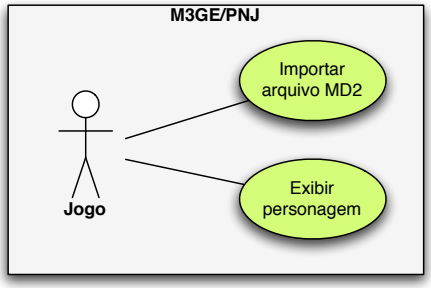


Figura 6: *Diagrama de casos de uso*

Para inserção do modelo 3D no grafo de cena, é necessário o uso de uma estrutura própria. Esta estrutura é apresentada como sendo o nodo *Mesh*, que representa um objeto 3D e armazena diversas características necessárias para sua manipulação através da API gráfica M3G. A estrutura armazena posição dos vértices, índices dos triângulos, textura, etc. A Figura 7 demonstra um diagrama de atividades que representa a criação de um nodo do tipo *Mesh*.

Devido ao fato de a aplicação (jogo) poder acessar tanto a M3GE quanto a M3G, optou-se por implementar o carregador de arquivo MD2 em um pacote de classes separado dessas arquiteturas. A M3G é utilizada para implementação da parte de visualização do modelo, porém o módulo é agregado ao motor de jogos M3GE. Para melhor distribuição das classes foram criados dois pacotes, o `md2loader` e o `datatypes`. O diagrama de classes dos dois pacotes é apresentado na figura 12.

A classe Md2Model é a responsável por montar um nó que será acrescentado ao grafo de cena. Este nó contém todas as especificações necessárias para se desenhar o modelo existente. Pelo fato do arquivo MD2 ser dividido em duas partes, cabeçalho e

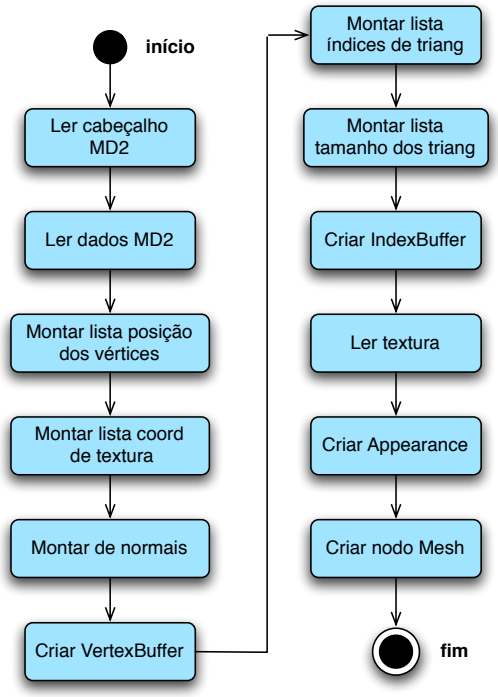


Figura 7: *Diagrama de atividades da criação do nodo Mesh*

dados, duas classes foram criadas, uma para cada parte do arquivo, sendo elas, a classe Md2Header responsável pela leitura e armazenamento do cabeçalho do arquivo, e a md2Data responsável pela leitura e armazenamento dos dados do arquivo. Para os vetores normais do modelo, existe uma lista pré calculada, que está presente na classe Md2Normal. Ainda para a leitura dos dados dos arquivos teve-se que utilizar uma classe para o tratamento dos tipos de dados, chamada MD2LittleEndianDataInputStream. Esta classe faz parte do projeto Xith3D [Lehmann 2004]. Este projeto é um pacote para tratar cenas gráficas e fazer sua renderização. Ele é todo escrito em Java e foca a criação de jogos.

As classes pertencentes ao pacote `datatypes` são utilizadas para armazenar em objetos todo conteúdo proveniente da leitura dos dados. Desta forma pode-se trabalhar com uma estrutura de dados própria para o arquivo MD2. A classe Md2Vect3 é responsável pelo armazenamento de um vetor no espaço cartesiano tridimensional com as três coordenadas: x, y e z. A classe Md2Skin é responsável pelo armazenamento do nome da textura do modelo. A classe Md2TextCoord é responsável pelo armazenamento das coordenadas da textura através dos atributos s e t. A classe Md2Triangle é responsável pelo armazenamento de um triângulo através de três índices dos vértices e três índices das coordenadas de textura. A classe Md2Vertex é responsável pelo armazenamento de um vértice através de três coordenadas e um índice para o vetor normal. A classe Md2Frame é responsável pelo armazenamento das informações dos quadros de animação do modelo, compostos por um vetor com o fator de escala, um vetor de translação, o nome do quadro e uma lista dos vértices. A classe Md2GlCmd é responsável pelo armazenamento de uma estrutura para ser utilizada com primitivas OpenGL quando renderizadas. Esta estrutura guarda apenas as coordenadas de textura s e t, e um índice para o vértice.

A figura 8 mostra a estrutura da M3GE após a agregação do módulo para suporte a PNJs com arquivos MD2. Como a implementação é feita em um pacote separado dos três módulos básicos da M3GE, única alteração feita no antigo código fonte da M3GE foi a adição do método `addNPC()` à classe EngineCanvas. Através deste método é possível adicionar um modelo MD2 ao grafo de cena.

Para a leitura de um arquivo binário optou-se por utilizar a classe DataInputStream que implementa a interface DataInput da J2ME pelo fato dela já disponibilizar diversos métodos de retorno nos tipos de dados Java. Porém, esta classe acaba invertendo a seqüência de *bytes*. Para contornar este problema é utilizada a classe

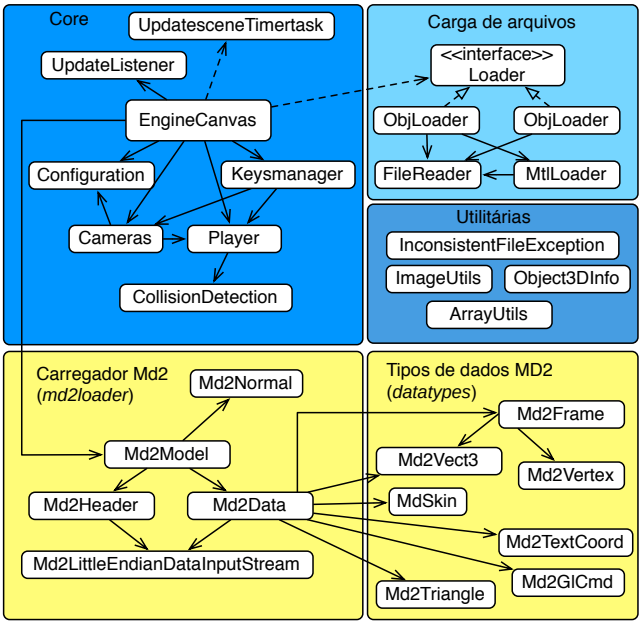


Figura 8: *Carregador MD2 inserido na M3GE*

MD2LittleEndianDataInputStream que sobrescreve os métodos da classe DataInputStream fazendo a mudança na seqüência dos *bytes* quando há dois ou mais *bytes* para serem lidos. Um exemplo desta conversão pode ser analisado no código abaixo, onde é feita a leitura de quatro *bytes*. Após isso, sua seqüência é invertida e retorna-se o valor no tipo de dado primitivo de Java, o int, que representa um número inteiro.

```
public int readInt() throws IOException {
    int[] res=new int[4];
    for(int i=3;i>=0;i--)
        res[i]=din.read();
    return ((res[0] & 0xff) << 24) |
           ((res[1] & 0xff) << 16) |
           ((res[2] & 0xff) << 8) |
           (res[3] & 0xff);
}
```

O gerenciador gráfico da M3GE já implementa funções capazes de desenhar os objetos em cena. Então para o PNJ ser desenhado, basta anexá-lo ao grafo de cena. Porém, para isto ocorrer o modelo deve ter uma estrutura compatível o nodo *Mesh* pertencente à biblioteca M3G. Este nodo define um objeto 3D através de seus triângulos juntamente com suas características conforme demonstrado na figura 9.

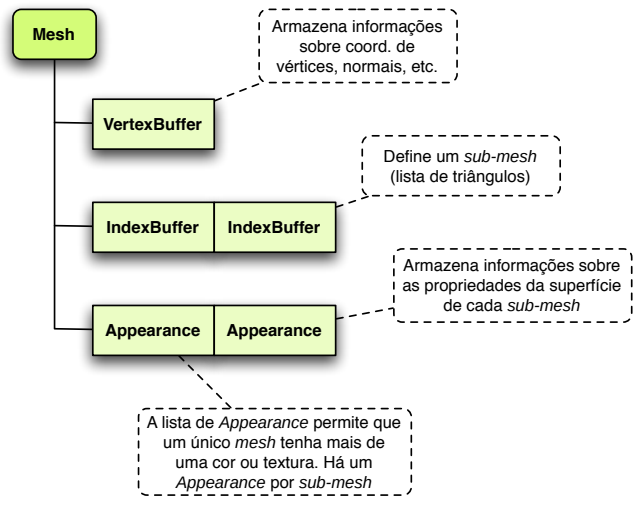


Figura 9: *Estrutura do nodo Mesh*

O nodo *Mesh* tem em sua estrutura o VertexBuffer que armazena informações relativas a posição dos vetores, a lista de normais, a lista de cores e as coordenadas da textura. Ainda possui uma lista de IndexBuffer que define os triângulos. Outra parte presente na estrutura é a lista de Appearance que é composta pelas informações de luzes, pelo modo de desenho do polígono, o modo de composição dos *pixels*, o modo de visualização baseado na distância da câmera e por último uma lista de texturas.

A classe Md2Model tem como função fazer as chamadas de leitura do arquivo MD2 e, a partir dos dados recebidos, montar uma estrutura completa de um nodo Mesh capaz de ser desenhado pela M3GE. Isto ocorre a partir do construtor da própria classe que recebe como parâmetros o local e nome do arquivo MD2, o local e nome do arquivo de textura e o índice do quadro a ser visualizado. Os métodos de leitura de arquivo MD2 desta classe convertem e preparam os dados de acordo com a estrutura do nodo *Mesh*, que é então armazenado no atributo model do objeto instanciado pela classe Md2Model. Em seguida ele é adicionado ao grafo de cena através do método `addNPC()` da classe EngineCanvas. Este método foi projetado para receber quatro parâmetros: local e nome do arquivo MD2, local e nome do arquivo de textura, número do quadro a ser visualizado e o nome do modelo. O primeiro passo deste método é criar um nodo Mesh através da criação de um objeto da classe Md2Model responsável pela leitura e estruturação do modelo. Em seguida é utilizada a classe Object3DInfo, que já estava implementada na M3GE, para inserir um nome ao modelo. Este nome terá utilidade para o dispositivo de tiros implementado na M3GE, podendo assim distinguir qual objeto foi atingido. Após isso, basta inserir o nodo *Mesh* a um grupo e depois incluí-lo ao nodo especial *World* que armazena todo o grafo de cena do jogo. O nodo *World* é um atributo da classe EngineCanvas.

9 Resultados

Para desenvolvimento de todo o projeto, incluindo os exemplos, foi utilizada a IDE Eclipse, com J2ME, Sun Java Wireless Toolkit, as APIs M3GE e M3G, e o Eclipse ME. Durante a implementação, vários testes foram realizados, inclusive avaliando resultados em diversos tipos de visualização. A figura 1, por exemplo, ilustra a visualização apenas com coordenadas de vértices e triângulos, com vetores normais, e com texturas, respectivamente. Nesta terceira visualização pode-se ver melhor os detalhes do personagem através do *skin* escolhido para ser utilizado como textura, cujo arquivo está no formato PNG.

Figura 10: *Jogo exemplo com importação de PNJs*

Para finalizar os testes e verificar a viabilidade do projeto foi implementado um protótipo de jogo, ilustrado na figura 10. Este protótipo é baseado no antigo jogo exemplo da M3GE, porém nesta versão foram adicionados três personagens a cena: o Knight

(arquivos knight.md2 e knight.png), o Centaur (arquivos centaur.md2 e centaur.png) e o Knight Evil (arquivos knight.md2 e knight_evil.png). Os personagens Knight e Knight Evil são carregados a partir do mesmo arquivo MD2 porém são utilizados skins diferentes. Toda funcionalidade que já estava presente no antigo jogo exemplo continuou funcionando durante e após a inclusão dos personagens. Até mesmo o tratamento de colisão que já estava implementado pela M3GE funcionou perfeitamente nos personagens adicionados. Pode-se inclusive utilizar a funcionalidade de tiro presente no jogo, observando as mensagens de texto exibidas quando algum PNJ é atingido.

Uma deficiência inicial constatada foi a impossibilidade de carregamento de mais de um quadro do MD2, devido à pouca quantidade de memória *heap* padrão disponível tanto no emulador quanto no dispositivo real onde foram feitos testes. Embora o suporte à animação esteja implementado e tenham sido feitos testes com até 5 quadros de animação para o modelo Knight, invariavelmente surge exceção de falta de memória. De certa forma isto já era esperado no início do projeto, entretanto seguiu-se a premissa de que brevemente os telefones celulares rodando Java terão memória suficiente para execução de jogos maiores e mais complexos.

Para avaliação do problema de quantidade disponível de memória foi utilizado o monitor de memória do Sun Java Wireless Toolkit. Constatou-se que quando ocorre a exceção "Out of memory", ela é realmente devida ao consumo de memória por parte das classes instanciadas. A configuração padrão do emulador possui cerca de 500 Kb de memória *heap*, compatível com a quantidade existente na maioria dos celulares atualmente disponíveis no mercado. Porém alguns celulares mais novos já possuem *heap* de 1.5 Mb. Percebeu-se também que o problema não está exatamente no armazenamento do modelo e sim na complexidade do procedimento de importação que demanda a utilização de diversas classes até que se tenha o nodo Mesh pronto para ser desenhado. Na verdade, para a leitura dos personagens foram utilizados mais de 500 Kb, porém antes que o *heap* chegasse ao seu limite alguns objetos instanciados que não iriam ser mais utilizados eram descartados pelo *Garbage Collector* do Java.

Ainda destaca-se a demora na leitura e visualização do modelo que leva em torno de trinta segundos para aparecer na tela do celular, acarretando em um tempo total para a carga completa do jogo em torno de um minuto e quarenta e cinco segundos. Isto pode ser considerado um fato negativo, pois alguns jogos podem necessitar que seus PNJs sejam apresentados instantaneamente na cena, e o problema se agrava quando há a necessidade de apresentar mais de um personagem simultaneamente. Nos testes realizados em um aparelho celular Siemens CX 65, a renderização ocorreu em 5 *frames* por segundo (fps).

Tabela 2: *Testes de consumo de memória*

Modelo	Weapon	Axe	Centaur	Knight
Tam MD2 (Kb)	38.0	62.0	240.0	312.0
Tam PNG (Kb)	42.8	7.29	157.0	132.0
Num Text Coord	70	78	469	307
Num Triang	72	126	536	634
Num Frames	173	173	199	198
Mem 1 Frame	84.5	57.0	276.0	263.9
Mem 5 Frames	93.5	72.6	322.2	338.5
Mem all Frames	471.9	732.8	3053.9	3942.4

Alguns testes adicionais foram realizados com o emulador, configurando-se sua memória *heap* no valor máximo permitido de 65536 Kbytes. A tabela 2 apresenta os resultados de consumo de memória em Kb de quatro modelos MD2 diferentes, com exibição e apenas um quadro de animação (*Mem 1 Frame*), cinco quadros (*Mem 5 Frames*) e finalmente todos os quadros (*Mem all Frames*). Os testes consideram o tamanho do arquivo de modelo MD2 (*Tam MD2*) em Kb, o tamanho do arquivo de texturas (*Tam PNG*), as quantidades de coordenadas de textura (*Num Text Coord*), triângulos (*Num Triang*) e quadros de animação (*Num Frames*). Para todos os casos, a taxa de renderização ficou em torno de 15 a 17 fps. Na figura 11 são exibidos *screenshots* dos 4 testes. Os resultados mostram um consumo de memória ainda proibitivo, porém passível de se tornar viável com o aumento de memória disponível nos futuros modelos de telefones celulares.

Atualmente a motor de jogos wGEN possui mais funcionalidades implementadas do que a M3GE (tais como recursos de áudio, por exemplo). Porém a M3GE acaba ganhando a possibilidade de importar arquivos MD2, coisa que não é possível na wGEN, além do fato de a wGEN ser eminentemente um motor para jogos 2D. Devido ao fato da wGEN ter sido implementada utilizando a J2ME, acredita-se que caso futuramente ela venha a ter recursos 3D, poderá com relativa facilidade incluir modelos MD2 utilizando a implementação feita no presente trabalho. Isto será bem simples, pois o presente projeto foi desenvolvido com características de um *plugin* à M3GE.

Figura 11: *Testes de consumo de memória*

O *framework* mOGE é implementado em um nível mais baixo, acessando diretamente a biblioteca OpenGL ES, diferentemente da solução adotada pela M3GE que trabalha com o M3G. No caso da M3GE, a M3G já traz muitas funcionalidades de visualização de objetos 3D em dispositivos móveis, que auxiliou no desenvolvimento do projeto. Por exemplo, o fato de poder incluir um nodo Mesh ao grafo de cena e a própria API M3G ser capaz de desenhar os objetos anexos a este grafo.

10 Conclusões

O presente trabalho apresentou um estudo e a implementação de recursos para inclusão de personagens não-jogadores na M3GE. Isto é feito através da leitura e visualização de modelos no formato M2D.

A visualização dos modelos apresentou boa qualidade gráfica, entretanto, conclui-se que utilizar o arquivo MD2 para adicionar

PNJ aos jogos é atualmente inviável no desenvolvimento de jogos comerciais perante a ainda pequena quantidade de memória disponível e à demora em renderizar os personagens nos atuais aparelhos de telefonia celular disponíveis no mercado brasileiro. Porém, acredita-se que este problema poderá ser contornado de duas formas. A primeira consiste na tendência de melhoria nos recursos de hardware dos futuros dispositivos móveis. A segunda consiste em utilizar técnicas de redução de nível de detalhe das malhas de polígonos dos modelos, diminuindo assim a carga de processamento gráfico exigida para sua renderização. Acredita-se que isto pode trazer bons resultados, e justifica-se devido à baixa resolução gráfica dos visores dos atuais aparelhos celulares.

Para testes e validação dos resultados, foi implementado um protótipo de jogo com mais de um PNJ em cena, entretanto, devido a limitações de memória dos dispositivos reais, não foi possível visualizar de forma satisfatória as animações dos modelos MD2. Foi possível a visualização de todos os quadros de animação nos modelos MD2 testados em emuladores, desde que estes fossem configurados para permitir o uso máximo de memória permitido.

Os resultados obtidos neste trabalho demonstram que é possível utilizar o arquivo MD2 para a manipulação de personagens em motores de jogos para dispositivos móveis, apesar da demora na importação dos modelos. Desta forma, abre-se a possibilidade de desenvolvimento de estudos que possibilitem novas descobertas para viabilizar a utilização de MD2 em telefones celulares, especialmente considerando as animações dos modelos como um requisito altamente desejável para o desenvolvimento de jogos com PNJs. Com esta primeira experiência, os autores sugerem também estudos para a integração de recursos de Inteligência Artificial aos PNJs, de forma a melhorar a qualidade dos jogos 3D para dispositivos móveis.

Referências

BYL, P. B. 2004. *Programming believable characters for computer games*. Charles River Media, Massachusetts.

CARANDINA, R., AND MARTINS, T. 2006. Suporte a jogos multiusuário em engine 3d para dispositivos móveis. Tech. rep., Centro Universitário Senac, São Paulo, Junho.

CONITEC CORPORATION, 2004. 3d gamestudio / a6. `http://conitec.net/a4info.htm`, Acesso em: 17 abr. 2006.

CRYSTAL SPACE, 2004. Crystal space 3d. `http://crystal.sourceforge.net`, Acesso em: 17 abr. 2006.

DE ALBUQUERQUE MACÊDO JR., I. J. 2005. moge – mobile graphics engine: o projeto de um motor gráfico 3d para a criação de jogos em dispositivos móveis. Tech. rep., UFPE.

ECLIPSE ENTERTAINMENT, 2004. Welcome to the home of genesis3d. `http://www.genesis3d.com/`, Acesso em: 17 abr. 2006.

ELEMENTS INTERACTIVE, 2007. Edgelib mobile game engine. `http://www.edgelib.com/`, Acesso em: 25 ago. 2007.

FINNEY, K. C. 2004. *3D game programming, all in one*. Thomson Course Technology, Boston.

FISHLABS ENTERTAINMENT GMBH, 2007. Fishlabs 3d mobile java games. `http://www.fishlabs.net/en/engine/index.php`, Acesso em: 25 ago. 2007.

GOMES, P. C. R., AND PAMPLONA, V. F. 2005. M3ge: um motor de jogos 3d para dispositivos móveis com suporte a mobile 3d graphics api. In *SBGames2005 - Simpósio Brasileiro de Jogos para Computador e entretenimento Digital, Anais do WJOGOS 2005*, SBC, 55–65.

GUILIANO, S., 2005. Autonomous productions. `http://www.autonomousproductions.com/`, Acesso em: 14 set. 2005.

HARRISON, L. T. 2003. *Introduction to 3D game engine design using directx 9 and C#*. Springer-Verlag, New York.

HENRY, D., 2004. Md2 file format (quake 2's model). `http://tfc.duke.free.fr/coding/md2-specs-en.html`, Acesso em: 24 mar. 2006.

HÖFELE, C., 2005. 3d graphics for java mobile devices, part 1: M3g's immediate mode. `http://www-128.ibm.com/developerworks/wireless/library/wi-mobile1/`, Acesso em: 17 mar. 2006.

KHRONOS GROUP, 2004. Opengl es: overview. `http://www.opengles.org/opengles/index.html`, Acesso em: 17 abr. 2006.

LEHMANN, J., 2004. Xith3d: contents. `http://xith.org/tutes/GettingStarted/html/contents.html`, Acesso em: 25 out. 2006.

MD2, 2005. Md2 - gpwiki. `http://gpwiki.org/index.php/MD2`, Acesso em: 24 mar. 2006.

MISFIT CODE, 2005. Misfit model 3d. `http://www.misfitcode.com/misfitmodel3d/olh_quakemd2.html`, Acesso em: 24 mar. 2006.

MÜLLER, L. F., FRANTZ, G. J., AND SCHREIBER, J. N. C. 2005. Qualcomm brew x sun j2me: um comparativo entre soluções para desenvolvimento de jogos em dispositivos móveis. In *SULCOMP - Congresso Sul Catarinense de Computação*, vol. 1, Sulcomp, 1–8.

NOKIA. 2003. Jsr-184 mobile 3d api for j2me. Tech. rep., [P.O.Box].

NOKIA, 2004. Ri binary for jsr-184 3d graphics api for j2me. `http://www.forum.nokia.com/main/`, Acesso em 19 abr. 2006.

OGRE3D, 2005. Ogre 3d: open source graphics engine. `http://www.ogre3d.org`, Acesso em: 15 jul. 2007.

PAIVA, F., 2003. O jogo das operadoras. `http://www.teletime.com.br/revista/55/capa.htm`, Acesso em: 05 abr. 2006.

PARALELO COMPUTAÇÃO, 2004. Fly3d.com.br. `http://www.fly3d.com.br`, Acesso em: 17 abr. 2006.

PESSOA, C. A. C. 2001. *wGEM*. Master's thesis, UFPE, Recife.

SÁNCHEZ, D., AND DALMAU, C. 2004. *Core techniques and algorithms in game programming*. New Riders, Indianapolis.

SANTEE, A. 2005. *Programação de jogos com C++ e DirectX*. Novatec Editora, São Paulo.

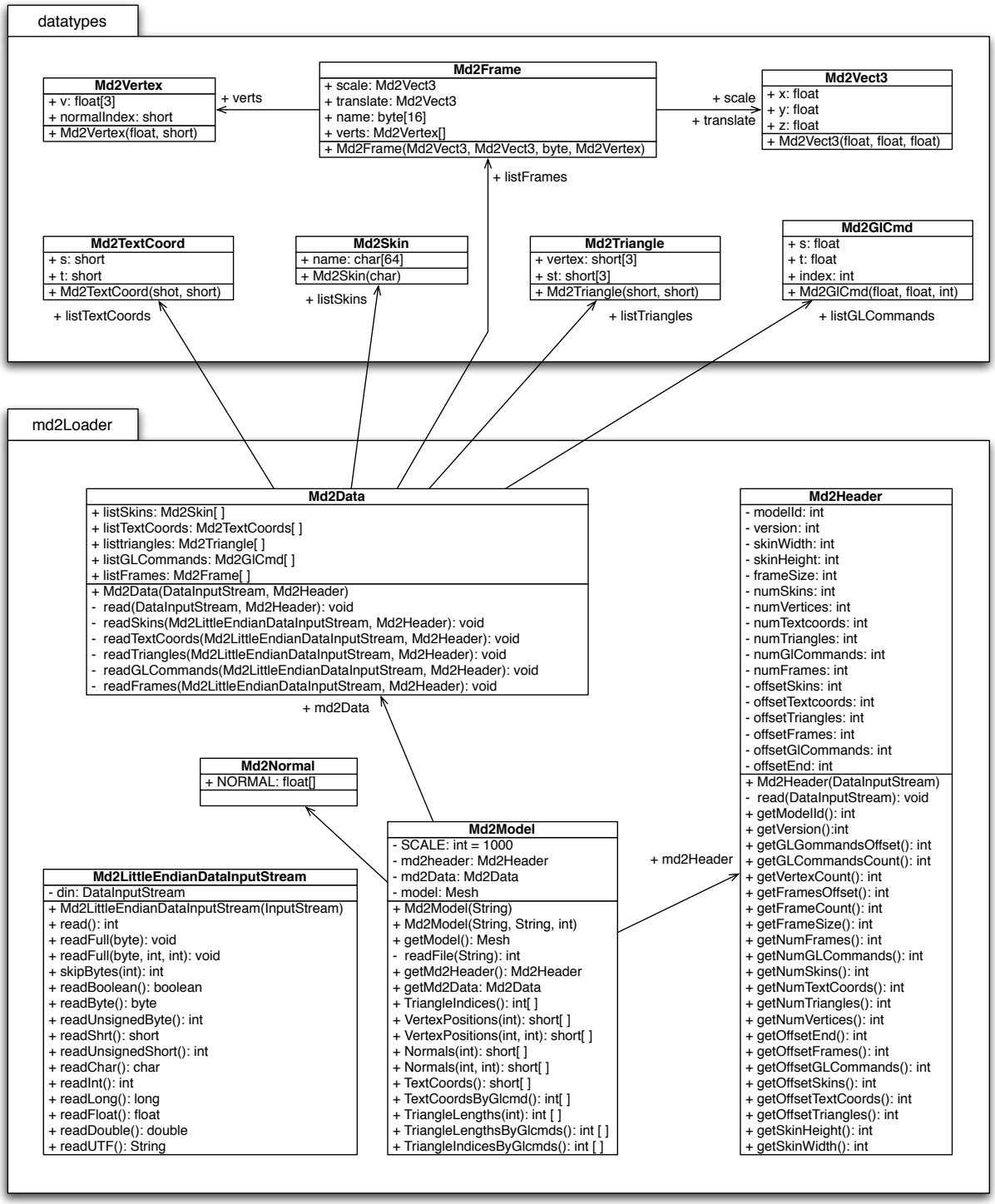


Figura 12: *Diagrama de classes dos pacotes md2loader e datatypes*

A Point-and-Shoot Technique for Immersive 3D Virtual Environments

Rafael P. Torchelsen, Marcos Slomp, André Spritzer, Luciana Porcher Nedel
Universidade Federal do Rio Grande do Sul

Figure 1. Users using the proposed point-and-shoot interaction technique. From left to right: a user wearing the virtual reality devices used by the technique; a close-up on another user performing the tasks; placing the 3D tracker on user's arm; a screenshot of the testbed application where the users performed the tasks.

Abstract

Modern computer games often explore graphics quality while leaving interaction and gameplay behind. Arcade machines may improve gamer experience by using special devices (instead of traditional joysticks) to interact with the game. We propose a technique for 3D “point-and-click” interaction that separates both camera and pointer control. It also explores audio and visual collision detection feedback to improve user spatial sense. The experiments were done within a virtual 3D environment, similar to that used in first person shooter games (FPS), a popular and well known genre by both the entertainment industry and gamers. It can also be used in applications other than games, to perform tasks such as “drag-and-dropping” objects, drawing and selecting in general. The results were compared with the traditional FPS interaction technique (mouse-driven camera with point-and-click interaction).

Keywords
Virtual Environment, Gameplay, Point-and-Click

Authors’ contact:
{rptorchelsen,spritzer,slomp,nedel}@inf.ufrgs.br

1. Introduction

A common problem with 3D interaction is finding a way to identify and interact with multiple objects at the same time [Lucas 2005]. In such applications, the user is often immersed in a virtual environment and must perform tasks while exploring. The point-and-shoot task exemplifies such a situation, in which the user must travel within the virtual world to seek and destroy targets that may be vulnerable to some specific action (such as different weapons). First-Person-Shooter games (FPS) are a traditional type of point-and-shoot application with these characteristics, being one of the most popular game genres, which includes titles such as the well known Quake and Unreal franchises.

One of the reasons that make FPS game so popular is the degree of immersion that the player experiences. Traditionally a camera is placed in the position of the eyes of the avatar to represent the point of view of a person in the game's world. The player can control the direction where the avatar is heading independently from the direction of view, which can also be controlled. However he cannot move the aim independently from the camera. Most FPS games mimic the behavior of a soldier that has a gun in his hands. For instance, if the weapon was a bazooka, the player would not be able to control the aiming independently from the camera. However, if it was a pistol, something like that is naturally not always true. This traditional gameplay also leads to another problem: when the player wants to move in a direction while shooting in another, he would not be able to see where he is going.

This work presents a technique where the camera control and target aiming are completely independent from each other and weapon switching can be performed quickly and easily. All of this is made intuitive the user by the mapping of his natural real word gestures to the actions of his avatar. A head-mounted display is used to dictate the direction of view, a pinch-glove for both shooting and weapon selecting and a 3D tracker for aiming. The shooting metaphor was chosen for the testbed application due to the popularity of FPS games and the intuitive mapping of real-world gestures to this type of virtual experience.

The technique works well for FPS applications and it can also be easily adapted for uses other than games, such as “drag-and-dropping” objects, drawing and selecting in general. Empirical results were presented to support the use of the technique as a means of improving user immersion and performance on switching weapons and avoiding “false hits”, as described in more details in Section 3.

Since the testbed application is very similar to a shooter game, this document starts with some background information and an overview of works describing related techniques. Next, the featured technique is explained in details, including its basic idea, hardware, software, experiment and data collection. This is followed by the results obtained and the problems encountered and pointed by the subjects

of the experiment. Finally, conclusions are summarized and a set of improvements, future works and research opportunities are listed.

2. Related Work

Despite the lack of publications from the game industry and developers, it is interesting to examine how shooter games emerged and evolved. After this briefing, some relevant related works will be discussed.

Arcade games were a very popular entertainment choice during the 80's. Missile Command [Web 1], a game from Atari, was launch in mid 1980 to become on of the greatest classic video games ever. The gameplay was very simple: move a cursor on the screen and press a button to shoot missiles at enemy missiles in order to keep them from hitting cities that were located below the screen. All interaction was done using a trackball and three buttons correspondent to three onscreen cannons that could be used by the player to accomplish his in-game mission. Within a level, the cursor moves always at the same speed, which is increased as the player advances in the game. The view was static, which means that the player was not free to explore the world. Years later, the same game was ported to Atari 2600, which introduced an axis-based joystick with a single button, but its gameplay barely changed.

Another relevant game appeared in 1984, when Duck Hunt [Web 2] was launched by Nintendo in Japan for its Nintendo Entertainment System (NES). The same idea from Missile Command was present here, but the player was able to interact with a more realistic metaphor: a pistol, which was used by pointing it at the screen and pressing its trigger. The screen was also static. Variants of the same game allows the player to interact using the default joystick from NES, which consists of a digital pad with four directions (DPad).

Many other games emerged from Missile Command and Duck Hunt, but, in 1992, Konami launched for the arcade the game Lethal Enforcers [Web 3], which uses the pistol metaphor and also allows limited weapon switching and camera movement. During gameplay, an icon of a weapon appears on the screen and if the user shoots at it, the current weapon becomes the one represented by the icon and it will be active until it runs out of bullets, in which case it will switch back to the initial gun. The camera moves horizontally (scrolling) after the player eliminates all enemies in an area (screen). VirtualCop [5], an arcade game by SEGA released in 1994, allowed the player to move in a 3D virtual environment, but the camera still moved without any interaction from the player. From this day on, no relevant modifications were made on the gameplay of this type of shooting game.

Two years later from VirtuaCop a new game style was born: Wolfenstein 3D [Web 4] was launched by id Software for the PC (DOS). The player was able to freely move within the virtual environments of the game, moving forward and backward and looking right or left. Looking up or down was not possible. A virtual gun was placed (drawn) on the bottom of the screen, centered, thus locking the aiming to the camera movement. Basically, all interaction was done with the traditional keyboard, including weapon switching. It was the beginning of the First-Person Shooter era, and many games derived from Wolfenstein 3D, including the well known Doom [Web 5] and Quake [Web 6] franchises, also from id Software.

The most significant advance in gameplay from Wolfenstein 3D to modern FPS gaming is the possibility of looking freely by using the mouse to control the camera. But again, aim and camera control are bound together. Some games, such as Freelancer [Web 7], from Microsoft, allowed aiming to be done independently from the camera, but since navigation and aiming were done using mouse, they tended to mutually interfere.

There were also FPS games that used virtual reality devices (head-mounted displays) to improve the gamer experience. In these cases, looking and aiming were again bound together and shooting and weapon switching was accomplished in non-intuitive ways, using special joysticks which were held by the player.

But outside of the game industry most of the work in techniques of 3D point-and-click [Ughini 2006] [Hugg 2006] [Jacob 1990] [Kyriakakis 2001] are related to direct selecting objects not predicting its trajectory.

Predicting the trajectory of objects is a common task to us and most of the time prediction of movement is mapped to the mouse [Huff 2006]. But we are used to track moving objects with the eyes, to take advantage of this work has being done to use both hands and eyes to select objects [Jacob 1990] [Yamato 2000] [Ware 2000]. Another important aspect in predicting movement is the sound produced in most of objects in movement, the sound can help locating objects and predict speed [Kyriakakis 2001].

Following the cited characteristics and work we present a technique to map the prediction of movements and selection of objects in a 3D immersive environment and compare it to a traditional mouse selection technique.

3. Method

Within this section, all implementation details will be discussed. First, an overview of the basic concept of the proposed technique is provided. The hardware available for the implementation of the concept is then presented, followed by the software used and implemented. The experiment is then discussed in details, including the hypothesis, dependent and independent variables, and what data was collected from the testbed application and from the questionnaire applied to each participant.

3.1 Concept

The technique works very well for point-and-shoot applications, since its goal is to skirt problems and limitations found in existing interaction techniques, the following items should be considered:

• user should be able to look freely;

• aiming independent from camera control;

• ability to switch between weapons;

• shooting.

It is important to perform all these actions naturally, i.e., the devices employed should allow intuitive and simple ways of mapping real world intentions and gestures to the virtual environment actions.

To implement the camera movements and garantees the immersion, a headmounted display with rotation tracking is used. For aiming, a single 3D tracker with only rotational degrees of freedom is sufficient. Shooting and weapon switching are performed using a data glove with contact sensors on each fingertip (pinch glove).

The user wears the head-mounted display an all of his movements are naturally mapped to the testbed application, which properly rotates the camera in accord with the user's real world movements. The user also wears a data glove on his dominant hand. When a contact between the thumb and any other fingers occurs, it means that the user wants to shoot and that the weapon of choice is the one correspondent to the finger that touched the thumb. The tracker is placed on the user's arm, since placing it on the data glove is not a viable option because hand movements may change its rotational state so that it points to a different place from where the user wants to aim at, which is not desirable (Figure 2).

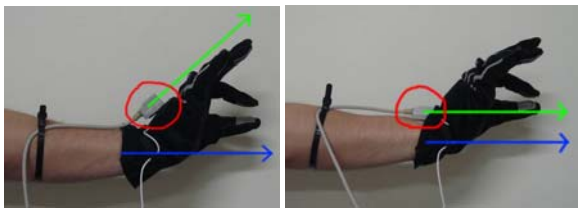


Figure 2. Tracker positioned on user's hand and arm, respectively (red circles). Blue arrows represent user's aim intention and green arrows represent tracker effective aim. It is preferable to place the tracker on his arm, since placing it on the glove may not match with user's aim intention.

Note that the user is not able to move freely (translation) within the environment, but as the main objective of the technique is pointing-and-shooting, this limitation also helps him to keep in mind his goal, avoiding distractions such as exploring the environment excessively and unnecessarily.

3.2 Hardware

The head-mounted display used was IISVR's VFX3D. The 3D tracker was Ascention's Flock of Birds and only its rotational features were used. The data glove used was Fakespace's Pinch Gloves. Figure 3 shows the virtual reality devices used. The computer used was a Pentium 4 1.4GHz with 512MB RAM, running Windows XP Professional. The video card was a GeForce Quadro FX2000.

This was the equipment chosen due to its availability at the laboratory where the technique was developed. As will be further mentioned, additional trackers and a more accurate head-mounted display may contribute positively to the success of the technique.

Figure 3. Devices used. Top-left: VFX3D (head-mounted display). Top-right: Pinch Gloves (data glove). Bottom: Flock of Birds (3D tracker).

3.3 Software

In terms of software, OpenGL was used for 3D rendering. The audio was implemented using OpenAL. A simple collision detection system was also implemented, dealing with boxes, spheres and planes, but box collision was not used in the testbed application. An exporter for the 3DS file format was implemented in order to import models created within 3D Studio MAX. Particles and textures were made using GIMP and Microsoft Paint.

The virtual environment thus provides much feedback to the user: particles, fog, radiosity-based illumination (using light maps), textured models, 3D sound effects, music and collision response. There are many virtual reality and 3D interaction applications, but few explore visually complex scenes. This is particularly important to this experiment since it can be essential for a higher immersion in the virtual world, which is one of the requirements of the experiment. The idea is to keep a simple and clean scene, without much details, but that still is visually pleasant to the user.

For device programming and communication, support SDK from manufacturers was used. Unfortunately, no helpful SDK or driver was found for the Pinch Gloves, so a driver for it had to be developed from scratch, based on code fragments found from a few sources on the Internet.

More details about the design of the virtual environment and the user tasks are described in the next subsection.

4. Evaluation

To validate the technique, a testbed application was developed as well as two questionnaires, one to be

answered before the user experience on the testbed and another questionnaire afterwards.

4.1 Experiment

The testbed application consists of a simple first-person shooter game, that was codenamed GhostHunt. The user must seek for ghosts (targets) and destroy them using his weapons. When all ghosts of an area are destroyed, the camera moves (translates) to another area, where a new gate appears and more ghosts emanate from it. Four types of weapons were made available: yellow, green, blue and pink. Ghosts appear in these same four colors. Every time a ghost gets hit by a shot of its color ("true-hit"), its color randomly changes to one of the other weapon colors. After four hits, the ghost is destroyed. If a shot from a different color hits a ghost, the "false-hit" counter is incremented. Refer to Figure 4 to some screenshots of the testbed application.

The user task is quite simple: clear all ghosts in the area using his weapons and finally face a "boss". Ghost size and initial color are chosen randomly. They move using route points within an area (near its spawn gate). When a route point is reached, it randomly chooses another one and selects a new linear and angular velocity. The boss acts just like the ordinary ghosts, but instead of being completely destroyed after four hits, each time it gets a true-hit, it is destroyed but spawns two "mini-bosses", which are identical to the boss, but smaller in size. The same applies to these mini-bosses and this goes on until no additional spawns are permitted.

Audio feedback is provided, using 3D sound effects to help the user locate the ghosts that are outside his field of view. Music was added to help cutting off the user from the real world, enhancing the immersion sense.

The user had to clear the level twice: one using the traditional interaction technique of first-person shooters, using the keyboard (for weapon switching) and mouse (looking, aiming and shooting), and another using the devices and the presently proposed interaction technique.

Before the experiment, the user was given the opportunity to practice in both modes. A simple launcher program was developed to collect user information (such as identification and dominant hand) and create configuration files that were used as input by the GhostHunt application. The launcher also randomly sorted the order of the practice sections as well as the order of the experiment sections. Figure 5 shows a user interacting with the environment using the technique described in this document.

4.2 Hypotheses

When comparing the new interaction technique with the traditional first-person shooter, the following items are supposed to be verified:

- **H1**: it will be easier to seek enemies using the head mounted display;
- **H2**: it will be easier to aim and hit enemies using the pinch glove;
- **H3**: weapon switching will be easier and faster using the pinch glove;
- **H4**: performance will be better with the independent aiming and looking provided by the VR devices.

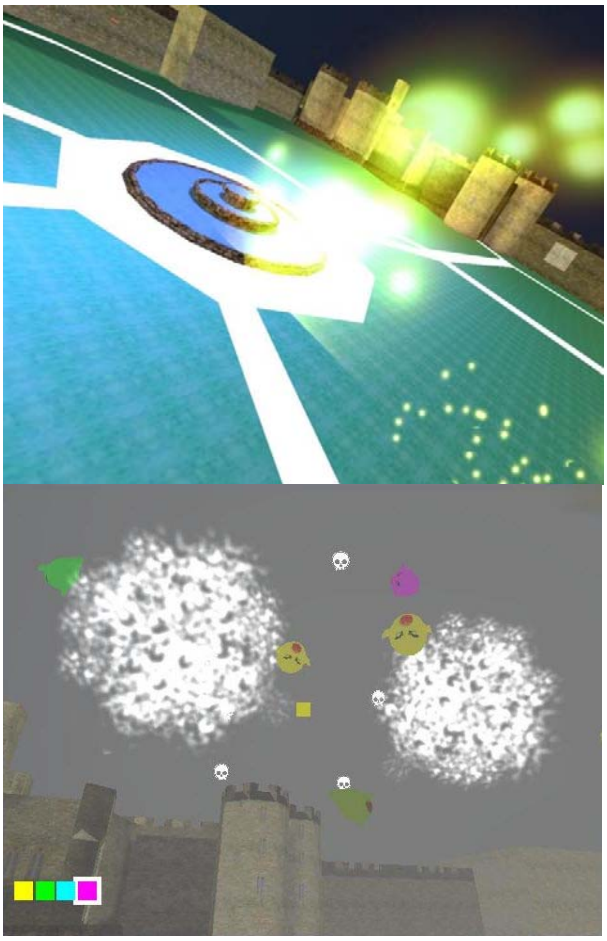


Figure 4. Screenshots from the testbed application. In top, an overview of the virtual environment used, with some particle effects, without fog. At bottom, a screenshot of an in-game experience (fog is enabled): the yellow square near the center of the picture represents the aim of the user; the squares at the bottom of the picture represents the weapons (white border marks the selected weapon); ghosts appears in different colors, spawning from gates (white clouds); white skulls are particles emitted when a true-hit occurs.

4.3 Independent and dependent variables

Only one independent variable is tested: performing the task using the traditional interaction technique and the new one, in which camera rotation and aiming are independent from each other. For the dependent variables, the following items were considered: time to complete the task, "true-hit" ratio, "false-hit" ratio, immersion level (subjective), fun level (subjective).

After the tests, when analyzing the resulting logs and the feedback given by the users, many interesting facts were noticed and are described in more details in Sections 5, 6 and 7.

Figure 5. User playing GhostHunt with the proposed technique: he is wearing a head-mounted display, a pinch glove and a 3D tracker on his arm.

4.4 Data Collection

The log file resulting from the experiment contains the following information:

• clear time: time spent to complete the task;

• shots: total number of shots fired;

• false hits: shots that collided with a target but are not in the same color as it;

The launcher application also asks the user what is his dominant hand before calling the practices and tests.

All participants were instructed to answer a questionnaire, with pre and post experiment questions. It was composed only from simple multiple-choice questions, helping the user to be as direct and clear as possible on his answers.

At the pre-test, the following was asked to the users: name, age, sex, level of education, video-game experience, first-person shooter games experience and virtual reality devices experience.

Where the experience was measured with four levels: 1-None; 2-Poor; 3-Medium; 4-High.

For the post-test, the following questions were asked to the subjects, comparing the traditional FPS experience with the new technique:

• Which one was more fun?

• Which one was more tiring?

• In which one was it easier to seek enemies?

• In which one was it easier to aim and hit?

• In which one was it easier to switch weapons?

• Which one was more immersive?

• In which one the sound was more helpful?

• Was independent looking and aiming helpful?

A subjective and optional question was provided at the end of the questionnaire, in which the subjects could write their comments about the experiment. The information collected from these notes proved to be very valuable to confirm some already expected behaviors and problems. This is described in more details in Section 5.

5. Analysis of the Results

To validate the hypotheses, the questionnaires and logs from the tests were analyzed, resulting in the following:

H1 - It will be easier to seek enemies: The opinion of the test subjects confirms the hypothesis. Figure 6 shows that 60% of the testers prefer the head-mounted device to search for the ghosts;

H2 - It will be easier to aim and hit enemies: The opinion of the testers and the logs refutes the hypothesis. From the questionnaire we can see that 85% prefer the mouse to aim and from the logs we can see that the hit-ratio (Figure 7) is better in the traditional simulation;

H3 - Weapon switching will be easier and faster: The questionnaire confirm the hypotheses with 85% of the testers preferring the Pinch Glove to switch amongst weapons;

H4 - Performance will be improved with independent aiming and looking: The hypothesis can not be confirmed because the opinion of the testers was divided, with 50% preferring the new method and 50%, the traditional. The time to complete the task suggests that the hypothesis is not valid. Figure 7 shows that the time taken to complete the tasks is increased by 10% when the devices are used.

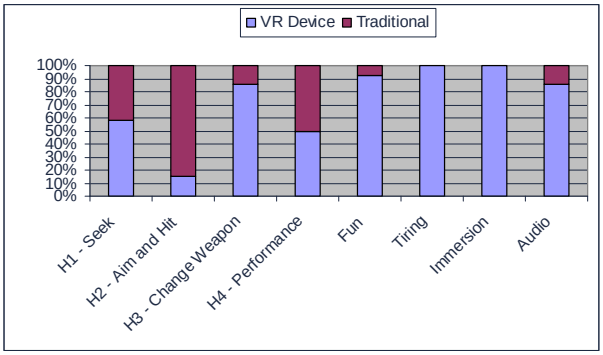


Figure 6. Results for the hypothesis and questionnaire.

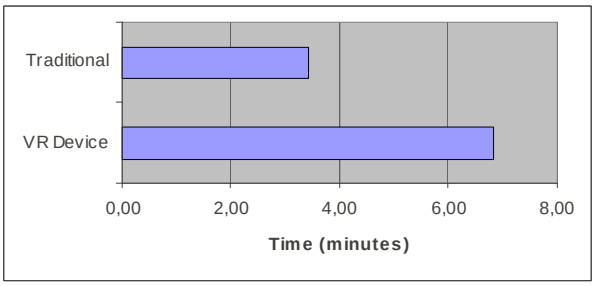


Figure 7. Time taken to complete the task.

The increase of time to complete the task is linked to the fact that about 10% of the testers had never used virtual reality devices in the past. Figure 8 illustrate the

experience of the testers with games, FPS games and virtual reality device.

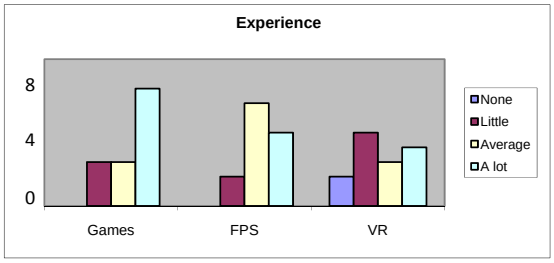


Figure 8. Past experience of the testers. We had a total of 15 testers.

As to the amount of shots fired and the amount of true hits, Figure 9 shows that the traditional simulation generates 16% more true hits. This data is linked to the fact that most of the users had a lot of experience with FPS and little with virtual reality devices.

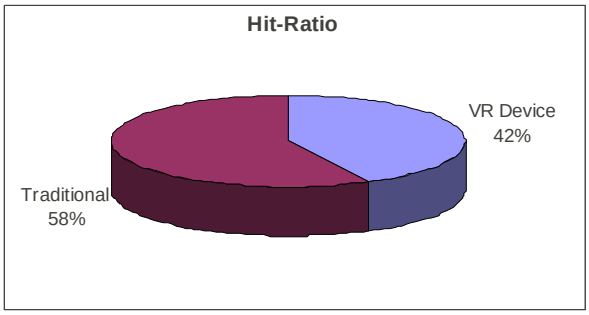


Figure 9. Hit Ratio.

An interesting data collected in the logs and illustrated in Figure 10 is that there were 32% more false hits in the traditional simulation. It indicates that the selection of weapons using the virtual reality devices is more efficient.

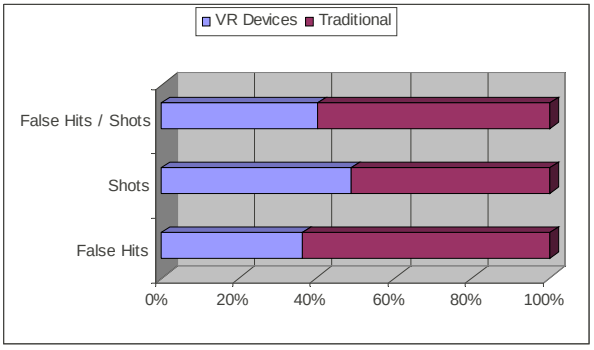


Figure 10. Shots and false hits data collected from the logs.

Another data that indicates that the presented technique is promising is that 100% of the testers affirmed that it is more immersive and 90% indicated it is more fun.

Moreover, the opinion of the testers as to the audio is that it is more helpful in the presented technique. In their opinion, it makes the process of seeking the ghosts with the head-mounted display easier (since 3D audio was used).

It is an unanimous opinion, though, that the intense use of the head-mounted display and the arms make the technique physically tiring. An interesting result was that the fatigue was only noticed by the testers after the end of the test – while they were playing, they didn't feel tired. This is a probable result of the test being entertaining and immersive, which are goals of the proposed technique.

6. Feedback and Problems

During the development of the testbed application, several issues were noticed: difficulties of device calibration, problems when converting rotational data between the device driver and the application, rendering fill rate problems when a large amount of particles appears at the same time on the screen, the limited resolution of the available head-mounted display and electromagnetic interference on the 3D tracker. A lot of time was spent adjusting parameters and making conversions and data wrapping to try to minimize some of them. More time was spent on this due the lack of software for the Pinch Gloves, for which a new driver had to be developed.

The participants (a total of 15 users) feedback notes have proved themselves to be a really valuable source of information. They helped confirm many speculations that were raised during the development, when the problems previously noticed and described above were faced. Below there is a list of the most relevant user feedback that was given after the experiments, followed by the authors' comments about it:

• **It was difficult to see the aim when using the head-mounted display**: Nothing much can be done about this, since the available head-mounted display has a very poor resolution, field of view and color quality.

• **The aim color could be changed according to the selected weapon:** Very easy to implement, despite not being common in first-person shooter games.

• **The aim style could be changed**: A simple yellow square was used for the aim, but it can be easily replaced by any other texture.

• **Lack of practice and contact with virtual reality devices**: Since virtual reality is not very popular amongst people in general, there is not much that can be done about this, but more training sections may improve user performance.

• **Shots are too slow**: The speed can be increased, but we are not convinced this is a real problem, since just one subject pointed this out.

• **Slowdowns while playing**: It happens when many particles appears on the screen at the same time, causing a fill rate problem. One option is to limit the number of simultaneous particles, since many of the users tended to use their weapons like a machine gun.

• **The Pinch Gloves failed to respond sometimes**: It was noticed before that the Pinch Gloves do not fit very well in all hand sizes, which can cause difficulties in detecting the contact between the fingers – mainly when trying to touch the thumb with the little finger.

• **Aiming problems when the hand is too far away from the body**: The data retrieved from the tracker presents an increasing and significant error as the tracker is moved away from its magnetic base. This error is accentuated by the presence of nearby metal objects, which may cause a magnetic interference.

With more appropriate devices and more calibration and training, the authors believe it is possible to highly increase user performance and validate the second hypothesis. Despite all these problems, all users preferred the new technique, considering it more fun and immersive. Another interesting fact is that the weight of the head-mounted display was barely noticed when doing the tests of the new technique, and the authors believe that the immersion level provided by the devices and the feedback given by the testbed application helped the user to ignore its weight.

7. Conclusions and Future Works

Since it has been some time that first-person shooters use the same type of interaction, the proposed technique comes to revigorate the genre. To that goal the proposed point-and-shooting technique presents positive results and feedback. The results showed that weapon switching was more entertaining and efficient with the proposed technique. It was also noticed that users tended to perform less false-hits using the new method. It received mostly positive feedback from testers, especially in regards to the entertainment level and the mapping of real-life movements. Such mapping can be used not only in FPS games, but also in any application that requires fast switching of tools, selecting, point-and-clicking, etc.

From the results obtained with the experiment, it can be said that the presented technique allows for a more immersive and entertaining interaction with a 3D environment, to a point that the fatigue felt by the users due to the head-mounted display and the arm movements necessary to control the aiming passed unnoticed by the subjects until after the tests were over.

With the feedback given by the participants and due to problems encountered during the development stage, a series of improvements can be made to improve user performance and then, possibly, validate the second hypothesis.

Most of the request was related to graphics, devices and task:

Graphics: Improve the graphics engine to avoid slowdowns and include extra scenes were the most requested improvements.

Devices: A common request was for a rotatory chair to decrease the fadigue; add haptic and wind feedback; command the experiments in a room isolated from any metal that can interfere with the 3D tracker; use more than one tracker to estimate the user's aim; develop a “bracelet” to comfortably integrate the tracker and pinch gloves.

Task: Training some users for some time using the new technique and confront them with users using the traditional interaction; develop tasks that need the use of both hands; study the possibility of adding translational degrees of freedom to allow the user to move and interact with the virtual environment; add offensive actions for ghosts to study user's reactions.

8. Acknowledgements

The authors would like to thank the CNPq for the support and all the volunteers who participated in the experiment.

9. References

LUCAS, J.F., BOWMAN, D.A., CHEN, J., and WINGRAVE, C.A., 2005. "Design and Evaluation of 3D Multiple Object Selection Techniques," Proc. of the ACM I3D, March 2005.

DELIGIANNIDIS, L. and JACOB, R.J.K. 2006. "The VR Scooter: Wind and Tactile Feedback Improve User Performance". IEEE Symposium on 3D User Interfaces 2006 (3DUI 2006). March 2006. pages: 143-150.

UGHINI, CLEBER S.; BLANCO, FAUSTO R.; PINTO, FRANCISCO M.; FREITAS, CARLA M.D.S.; NEDEL, LUCIANA P. 2006. “*EyeScope: A 3D Interaction Technique for Accurate Object Selection in Immersive Environments*”, In: SVR 2006 - SBC Symposium on Virtual Reality, 2006, Belém: CESUPA, 2006. v. 1, p. 77-88.

HUFF, RAFAEL; SILVA ISABEL C.S.; FREITAS, CARLA M.D.S.; NEDEL, LUCIANA P. 2006. “*Seleção de Objetos em Ambientes Virtuais com Mouse 3D*”, In: SVR 2006 - SBC Symposium on Virtual Reality, 2006, Belém: CESUPA, 2006. v. 1, p. 349-360.

BOWMAN, D., KRUIJFF, E., LAVIOLA, J., and POUPYREV, I. 2005. 3D User Interfaces -Theory and Practice. Addison Wesley.

JACOB, R. J. K. 1990. What you look at is what you get: eye movement-based interaction techniques. In CHI ’90: Proceedings of the SIGCHI conference on Human factors in computing systems, pages 11–18, New York, NY, USA. ACM Press.

KYRIAKAKIS, C. 2001. Fundamental and technological limitations of immersive audio systems. In Jeffay, K. and Zhang, H., editors, Readings in multimedia computing and networking, page 0. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

NEDEL, L. P., FREITAS, C. M. D. S., JACOB, L. J., AND PIMENTA., M. 2003. Testing the use of egocentric interactive techniques in immersive virtual environments. In Interact 2003, pages 471–478. IFIP.

YAMATO, M., INOUE, K., MONDEN, A., TORII, K., AND ICHI MATSUMOTO, K. 2000. Button selection for general guis using eye and hand together. In AVI ’00: Proceedings of the working conference on Advanced visual interfaces, pages 270–273, New York, NY, USA. ACM Press.

WARE, C. 1990. Using hand position for virtual object placement. The Visual Computer, 6(5):245–253.

Web 1 http://en.wikipedia.org/wiki/Missile_Command (Visited on 14/12/2006)

Web 2 http://en.wikipedia.org/wiki/Duck_Hunt (Visited on 14/12/2006)

Web 3 http://en.wikipedia.org/wiki/Lethal_Enforcers (Visited on 14/12/2006)

Web 4 http://en.wikipedia.org/wiki/Wolfenstein_3D (Visited on 14/12/2006)

Web 5 http://en.wikipedia.org/wiki/Doom (Visited on 14/12/2006)

Web 6 http://en.wikipedia.org/wiki/Quake (Visited on 14/12/2006)

Web 7 http://en.wikipedia.org/wiki/Freelancer_%28computer_game%29 (Visited on 14/12/2006)

FURGBOL-PV: Um Ambiente para Realidade Mista Usando Futebol, Battle e Pac-Man

Silvia Silva da Costa Botelho
Eder Mateus Nunes Gonçalves
Gisele Moraes Simas
Rafael Gonçalves Colares
Renan Rosado de Almeida
Renan de Queiroz Maffei
Rodrigo Ruas Oliveira
Fundação Universidade Federal do Rio Grande - FURG

Abstract

This paper presents the usage of a CITIZEN mobile micro-robots in an environment of mixed reality in order to validate artificial intelligence and computational vision theories. Then we propose a flexible and modular architecture that enables autonomous robot control in coordinated tasks, like the soccer, the Pac-Man and the Battle games. We will present the aspects of the prototypes implementation.

Keywords:: Robot, Artificial Intelligence, Mixed Reality, Computational Vision.

Resumo

Este artigo apresenta a utilização de micro robôs móveis CITIZEN em um ambiente de realidade mista a fim de validar teorias de inteligência artificial e de visão computacional. Então, propõe-se uma arquitetura flexível e modular que possibilita o controle de robôs autônomos em tarefas coordenadas, como o jogo de futebol, o jogo Pac-Man e o jogo Battle. Aspectos relacionados à implementação do protótipo serão apresentados.

Keywords:: Robô, Inteligência Artificial, Realidade Mista, Visão Computacional.

Author's Contact:

silviacb@furg.br
eder@ee.furg.br
gisele@ee.furg.br
colares@ee.furg.br
renan@ee.furg.br
rqmaffei@ee.furg.br
rodro@ee.furg.br

1 Introdução

Atualmente, existem diversos tipos de robôs móveis como, por exemplo, veículos de resgate, de exploração espacial, de inspeção subaquática, dentre outros. Outra categoria de robôs abrange os robôs inteligentes, que podem ser utilizados em simulações ou até mesmo no entretenimento. Tal categoria de pequenos robôs móveis surge para atender um mercado até então suprido principalmente por simulação.

Além de atender esse novo segmento do mercado, a RoboCup [Kitano et al. 1997] introduziu em 2007 uma nova categoria proposta pela Citizen, denominada Physical Visualization (PV), que possibilita a fácil construção do ambiente virtual de simulação e a fácil validação de teorias relacionadas a robótica, inteligência artificial e visão computacional.

Paralelamente, a RoboCup, dentro da sub-liga PV, motiva os pesquisadores a planejar, evoluir e idealizar sistemas complexos, que misturam o real com o virtual, introduzindo o conceito de *Realidade Mista*. A técnica de *Realidade Mista* (RM) é uma variação particular da Realidade Virtual. Segundo [Tamura and Yamamoto 1998], a RM possibilita adicionar em ambientes reais, objetos gerados computacionalmente, fazendo um misto de real e virtual. A interação deve ser em tempo real, ou seja, qualquer alteração no mundo real será reproduzido no virtual imediatamente e vice-versa.

Conforme [Carvalho et al. 2007], RM pode ser descrita como um espectro no qual em um extremo está a Realidade Comum (sem alteração alguma), e no outro extremo está uma Realidade totalmente Virtual (RV). Ao longo desse espectro encontram-se a Realidade Aumentada (RA) e a Virtualidade Aumentada (VA). A RA é baseada no mundo real enriquecido com informações virtuais, enquanto a VA está baseada no mundo virtual enriquecido com informações do mundo real.

Desta forma, este artigo apresenta uma arquitetura genérica capaz de manipular desde agentes virtuais em jogos como futebol, Pac-Man e Battle City, até robôs reais utilizados em simulações em ambientes virtuais complexos. Também são descritos, de maneira sucinta, módulos encarregados da simulação e modelagem de objetos em ambientes virtuais. Além disso serão apresentados os módulos de visão, estratégia e comunicação responsáveis pela manipulação dos robôs no time de futebol de robôs FURGBOL-PV e nos jogos Pac-Man e Battle City.

2 Estrutura do PV

A sub-liga PV é uma categoria nova da RoboCup, caracterizada pelo uso de mini-robôs desenvolvidos pela Citizen, apresentados na Figura 1. Os mini-robôs, com dimensões de 1x2x1 cm, são alimentados por uma bateria de relógio. Esta categoria propõem criar robôs autônomos capazes de resolver problemas diversos em um ambiente de realidade mista criado por um simulador.

Figura 1: *Mini-robôs Citizen*

O ambiente proposto opera com o conceito de realidade mista. Nesta categoria, os mini-robôs Citizen atuam num

ambiente virtual projetado numa tela LCD. Uma câmera captura a imagem da tela e a posição dos robôs transmitindo-a a um computador que executa o simulador do ambiente. Os agentes conectados ao simulador obtêm suas percepções e decidem suas ações comunicando-as pela rede. As ações são recebidas pelos robôs por meio de um transmissor infravermelho, que executarão os comandos recebidos interagindo com o ambiente virtual, sobre a tela LCD. A Figura 2 a seguir ilustra a proposta do ambiente da sub-liga.

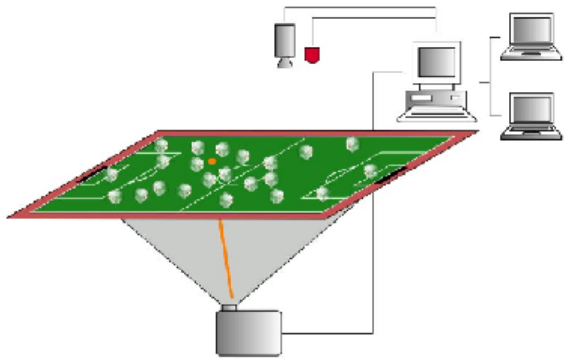


Figura 2: *Ambiente de realidade aumentada utilizado na sub-liga PV*

O objetivo principal da PV não é a construção dos agentes em si, mas o desenvolvimento e a otimização do ambiente. Não obstante, além da competição descrita acima, a PV propõe competições baseadas na melhoria do projeto dos mini-robôs (agregação de novos sensores, melhorias no firmware, etc.) e na proposta de novos ambientes utilizando a sua estrutura. Neste sentido, a PV se insere no FURGBol com o propósito de se tornar uma plataforma genérica para o desenvolvimento de Sistemas Multiagentes, não apenas no domínio do futebol, mas também em outros domínios a serem explorados. Deste modo, será possível desenvolver e implementar estratégias que serão testadas e verificadas em ambientes sob diversificadas restrições multiagente,porém sob a mesma estrutura.

Com essa estrutura é possível ter um ambiente de realidade aumentada. Esse ambiente oferece como vantagens:

- facilidade na construção de ambientes, não apenas futebol, mas também de ambientes difíceis de se reproduzir;
- flexibilidade na modificação das características dos objetos virtuais;
- redução do custo de projeto devido a emulação de sensores e atuadores.

3 O Time de Futebol de Robôs

A competição de futebol envolvendo os mini-robôs Citizen é realizada a partir de um plataforma padrão, baseada no paradigma cliente-servidor, disponibilizada pela RoboCup, apresentada na figura 3.

O servidor é responsável pela captura e pelo processamento das imagens. Nele é feita toda a segmentação de imagens e é devolvido aos clientes apenas as informações de posicionamento. Toda a simulação da partida, assim como a modelagem física da bola, também é implementada pelo servidor. A comunicação entre os robôs e o computador também é implementada pelo servidor e se dá através de um transmissor infravermelho. Para transmitir comandos infravermelho através do Linux é utilizado o pacote LIRC (Linux Infrared Remote Control).

O pacote LIRC permite a decodificação e o envio de sinais infravermelho. A parte mais importante dele é o `lircd`, que é um daemon responsável por receber os comandos do servidor e enviá-los aos mini-robôs por um transmissor conectado à porta serial.

Os clientes, cujas estratégias devem ser implementadas por cada um dos participantes, conecta-se ao servidor através de uma conexão UDP e passa a controlar os robôs. O cliente deve analisar os dados recebidos do servidor, decidir o comportamento dos robôs e enviar as ordens na forma de comandos ao servidor.

Para estabelecer a comunicação entre o cliente e o servidor, o aplicativo do cliente deve ser executado passando como parâmetros o endereço em que está rodando o servidor e a porta para o time do agente, designada previamente na configuração do servidor. Com isso, por intermédio de uma função de inicialização, é efetuado o registro do novo agente.

A etapa principal do programa dos clientes consiste em um laço que atualiza a cada ciclo os dados do servidor. O programa só sai desse laço quando não receber mensagens por mais de um segundo, sendo este considerado ausente e o cliente encaminhado para o encerramento.

Os dados são transmitidos, no laço principal, utilizando uma estrutura de um vetor polar (um módulo de uma distância e um ângulo). A posição do robô em relação a qualquer ponto do ambiente, como por exemplo a bola, o gol, ou companheiros de time, é obtida por meio de funções que calculam o vetor da distância do robô a estes pontos. É também nesse laço que as instruções para a movimentação do robô são enviadas pelo cliente, podendo-se indicar a direção do movimento ou a posição que se deseja ir. É ao fim de cada ciclo que todos os comandos infravermelho, são enviados de uma única vez pelo servidor, utilizando um transmissor baseado no pacote LIRC.

A estratégia utilizada para implementar o cliente do FURGBOL-PV é apresentada a seguir.

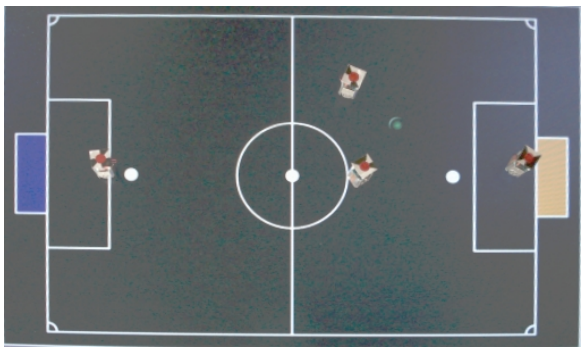


Figura 3: *Foto do jogo de futebol dos PVs*

3.1 Implementação do Cliente

A estratégia desenvolvida é implementada a partir de uma máquina de estados, cujos estados estão relacionados à posição dos jogadores e da bola, sendo as transições dadas em função da dinâmica do jogo, segundo a tabela 1.

Os robôs e também a bola são descritos por dois estados que identificam suas posições dentro do campo da partida: para o eixo x (que une ambas goleiras) podem ser áreas de defesa, ataque e meio; para o eixo y (perpendicular ao x), lateral esquerda, lateral direita e meio.

A Figura 4 mostra a estratégia do goleiro, que busca simular um goleiro real, usando o conceito de sair do gol para fechar o ângulo de chute do atacante. A idéia por trás dessa estratégia é diminuir o espaço aberto do gol para que o atacante adversário não tenha onde chutar a bola.

Para tanto, uma linha imaginária é desenhada na frente do gol, paralela a linha de fundo. Outra linha é desenhada do

Estado	Descrição	Ações
Disputa	dois robôs adversários estão perto da bola	Recuperar a bola
Posse	Somente membros do time estão perto da bola	Mover (com a bola)
Livre	Ninguém está perto da bola	Mover (Sem a bola)
Adversario	Somente robôs adversários estão perto da bola	Seguir um adversário
Gol	Bola etá nos limites do gol	Chutar

Tabela 1: *Os estados da bola e as ações do robôs*

centro da bola ao centro do gol. O nosso goleiro deve se posicionar na intersecção das duas linhas. A partir desta posição, outra linha imaginária é traçada perpendicularmente à linha do centro da bola ao centro do gol. O goleiro, movendo-se sobre essa linha percorre a mesma distância para desviar uma bola chutada tanto de um lado como do outro e ainda consegue diminuir essa distância.

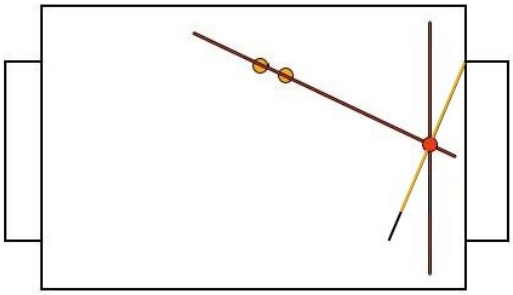


Figura 4: *Estratégia do Goleiro*

A estratégia de defesa usa a informação topológica da bola para mover o de robô de defesa. Cada área tem uma reta vertical que representa seu centro. Na Figura 5 nós podemos ver a linha central anterior à área da bola em direção ao gol da equipe. Essa estratégia calcula a intersecção entre esta linha e a linha central obtida a partir da posição atual e anterior da bola. O ponto de intersecção é onde o robô de defesa deve ir.

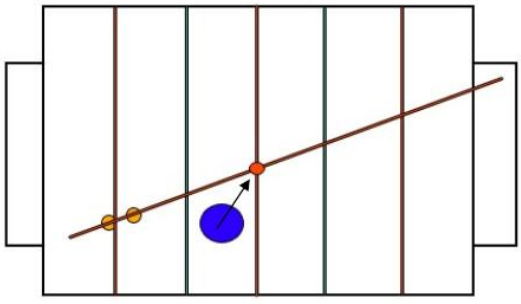


Figura 5: *Estratégia de Defesa*

A estratégia de ataque, Figura 6, é dividida em três estratégias, conforme o posicionamento da bola e dos robôs:

1. A primeira é aplicada quando o robô mais próximo da bola pode atacar. Ela ativa o drible e segue em direção ao gol adversário carregando a bola ou chuta em direção à este gol. Isso ocorre quando a intersecção entre uma reta que passa pelo centro da bola e do robô próximo a ela com a reta vertical do gol adversário resulta em um ponto dentro dos limites do gol.
2. A segunda estratégia de aproximação de ataque ocorre quando a intersecção descrita anteriormente resulta em um ponto fora dos limites do gol adversário. Nesse caso o robô deve se posicionar em uma posição válida para que ele possa carregar a bola para o gol. Essa posição pode ser obtida pela intersecção entre uma reta que liga o centro do gol adversário com a bola e uma circunferência de raio pré-definido centrada na bola.
3. A última estratégia de aproximação de ataque ocorre quando a intersecção da primeira estratégia de aproximação resulta em um ponto fora dos limites do gol adversário e a bola não está entre o robô e o gol adversário, como o que ocorreu na segunda estratégia de aproximação. Nesse caso é realizada uma intersecção entre uma reta vertical que passa pelo centro da bola e uma circunferência cujo centro é o centro da bola, de raio pré-definido. Quando o robô estiver localizado no ponto vermelho será realizada a segunda estratégia de aproximação, que tentará levar o robô finalmente para a primeira estratégia de aproximação, para assim carregar a bola em direção ao gol.

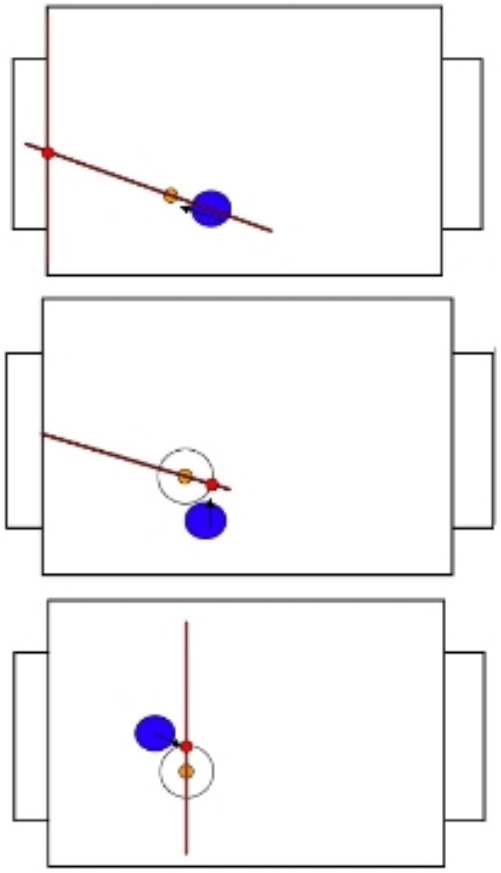


Figura 6: *Primeira, segunda e terceira estratégia de ataque, respectivamente*

Cada uma dessas três estratégias básicas determinam o alvo para o qual o robô deve se mover Nós usamos o *método de decomposição de células aproximadas* para alcançar cada alvo individual. Essa aproximação permite um planejamento da trajetória do robô evitando que ele colida. Esse método foi escolhido por ser simples, por fornecer uma decomposição uniforme e por ser fácil de implementar[Amato 2004]. O método de decomposição de células aproximadas como mostrado por [Latombe 1991] divide o campo em três células possíveis: vazias, cheias ou mistas. As células vazias não contém obstáculos. As cheias são completamente preenchidas por obstáculos. As células mistas são parcialmente preenchidas por obstáculos.

Então, cada célula vazia é conectada por um grafo à uma célula vazia vizinha. Depois é executado um algoritmo de me-

nor caminho usando Dynamic Programming Dijkstra [Cormen et al. 2001]. Neste grafo esse algoritmo fornece o menor caminho entre dois nodos oferecendo um planejamento de trajetória otimizado para cada robô, sem colisão.

Determinada a melhor estratégia, o software gera um vetor com os módulos de velocidade e direção para cada robô. Este vetor é transformado pelo cliente nos comandos do PV. O cliente por sua vez, envia os comandos para o Servidor, que os transmite para os robôs.

4 Implementação de um Ambiente de Simulação para Outros Jogos

O ambiente de simulação para os jogos propostos tem como base o paradigma utilizado na liga de simulação, incluindo aspectos do projeto de robôs reais e de realidade mista. Deste modo, está sendo desenvolvida uma arquitetura baseada em módulos, segundo a Figura 7.

Figura 7: *Arquitetura genérica do ambiente de simulação*

O ambiente simulação possui três módulos principais. O módulo Objetos Fixos que simula a posição dos objetos estáticos do campo, independente das ações dos robôs em cima do LCD. Esses objetos são desenhados uma vez na tela, e se mantém até o fim da simulação. As linhas de demarcação, paredes do ambiente encontram-se nessa classificação. O módulo Objetos Móveis, que representa os agentes, simula todos objetos passíveis de serem manipulados pelos robôs ou até robôs reais localizados remotamente e representados virtualmente no ambiente. Um robô é um exemplo de objeto móvel. E o módulo de Comunicação que envia aos robôs comandos de movimentação. Para implementar esse ambiente de simulação é necessário um enfoque mais profundo, no sentido de deixá-lo o mais real possível.

4.1 Descrição do Simulador Externo

Um dos principais benefícios dessa nova categoria da Robocup é a virtualidade. Essa virtualidade permite que partidas sejam disputadas remotamente, com um time em cada localidade, representados na televisão.

Para viabilizar jogos remotos foi desenvolvido o módulo Simulador Externo. Esse módulo se encarrega de estabelecer a comunicação entre um time remoto (cliente) e o servidor de simulação, que determina o estado do jogo. Entende-se por estado do jogo informações que descrevem a situação momentânea de cada uma das diferentes partidas, tais como: velocidade dos robôs, dos fantasmas, dos tanques de guerra e da bola. De posse dessas informações é possível simular em um ambiente virtual remoto a situação real dos robôs da equipe adversária.

A fim de promover essa comunicação foi desenvolvido em C++ um servidor de conexões UDP (Servidor Socket) que recebe informações da partida através da rede, simula o jogo, e envia informações referentes a simulação para os clientes.

4.2 Objetos Móveis

Os Objetos Móveis são aqueles que variam a posição no decorrer da simulação. Esse tipo de objeto é responsável pela representação dos robôs, dos fantasmas (no Pac-Man), além de outros. Para identificar os robôs são utilizadas duas manchas, a principal (amarelo ou azul), e outra secundária. Desse modo é possível diferenciar cada um robôs dos demais objetos projetados no LCD. O Módulo de Visão Global responsável pela identificação de todos objetos, fixos ou móveis, é descrito a seguir.

4.2.1 Módulo de Visão Global

Este módulo recebe um conjunto de frames advindos da câmera situada acima do campo, identificando os Objetos Móveis e fornecendo a posição e velocidade destes através de uma série de técnicas de processamento de imagens. Primeiramente, é aplicada uma correção de distorção radial, depois uma segmentação baseada em uma análise do espaço de cores HSV, e finalmente conhecimentos heurísticos são usados para localizar cada agente na cena.

Inicialmente as imagens capturadas estão no formato RGB(Red Green Blue). Buscando um sistema mais robusto a variações luminosas, transforma-se o espaço de cor para HSV(Hue Saturation Value) [Gonzalez and Woods 2001]. Desse modo é possivel definir classes de cores. A proposta desse módulo é classificar cada pixel do frame como pertencente a uma dessas classes. O processo inicia com a calibração onde o intervalo de H, S e V é definido para cada classe. Após, a imagem como um todo é classificada, pixel a pixel, pelo componente H. Mas como alguns intervalos de H são os mesmos para diferentes classes, se faz necessário utilizar outro componente, ou S ou V, de classificação, como citado por [Bruce et al. 2000].

A segmentação dos objetos é baseada na formação de manchas (blobs), que equivale ao número de pixels adjacentes de mesma classe. É definido um tamanho mínimo para as manchas, assim manchas menores são descartadas por serem provavelmente ruídos da captura.

O frame é percorrido procurando-se por manchas de duas classes principais, azul ou amarelo. Para melhorar a performance do sistema o frame é percorrido em janelas, não sendo necessário assim ler cada pixel do frame. O tamanho dessa janela é dependente do tamanho do campo e do tamanho do frame, sendo definido de forma empírica. Quando um pixel de alguma classe fundamental é encontrado então se calcula o tamanho da mancha pixel a pixel, ou seja, não se usam janelas.

No momento em que uma dessas manchas principais foi encontrada, é feita uma segunda procura num raio em torno do centro dessa mancha, tendo por objetivo localizar manchas de outras classes de cores secundárias, as quais servem para identificar o robô e sua direção. Essa mancha serve para determinar a direção e diferenciar o robô.

4.3 Comunicação

Os comandos enviados para os robôs são strings de 12 bits. Os primeiros 5 bits determinam o id do robô, os próximos 3 bits informam a velocidade e o sentido de giro da roda esquerda. Da mesma forma, os próximos 3 bits determinam a movimentação da roda direita. E finalmente, há um bit é

de verificação, sendo 1 caso a soma dos bits anteriores seja par e 0 caso contrário.

A versão atual do firmware do robô possui somente duas velocidades, rápida e devagar (a velocidade média ainda não foi implentada). A Tabela 2 mostra os comandos de controle dos robôs.

Além dos comandos padrões existe os comandos especiais, caracterizados pelos 3 primeiros bits ou, os últimos, todos 0. Na Tabela 2, os 4 últimos comandos são exemplos de comandos especiais. Esses comandos fazem o robô mover-se muito rápido por um curto período de tempo. Durante esse rápidos movimentos toda a CPU fica concentrada na tarefa de gerar o padrão de bits para controlar a velocidade dos motores. Comandos infravermelho não são escutados durante esse breve período.

Código	Código em 3 bits	Descrição
FF	001	Para frente rápido
FM	010	Para frente médio
FS	011	Para frente devagar
ST	100	Parar
BF	101	Para Trás rápido
BM	110	Para Trás médio
BS	111	Para Trás devagar
FNFN	000001	”Nitro”para frente
BNBN	000010	”Nitro”para trás
FNBN	000011	Giro no sentido horário
BNFN	000100	Giro no sentido anti-horário

Tabela 2: *Codificação Lógica de Sinais*

5 Utilização do Ambiente de Simulação implementado

5.1 O Pac-Man e Battle City

Um dos desafios da sub-liga PV é demonstrar a potencialidade da sua estrutura por meio de novas aplicações para os mini-robôs. Neste sentido, propõe-se a modificação dos jogos Pac-Man (Figura 8)e Battle City (Figura 9). No jogo do Pac-Man desenvolvido, o objetivo do jogador é comer todas as pastilhas e pílulas especiais que encontram-se no labirinto, porém existe fantasmas vagando que tentam captura-lo. Se o Pac-Man comer uma das pílulas especiais, ele ganha a habilidade de comer os fantasmas. No Battle city o jogador controla um tanque que atira e deve matar os outros tanques no cenário. Se um tiro acertar uma parede do cenário, esta será destruída.

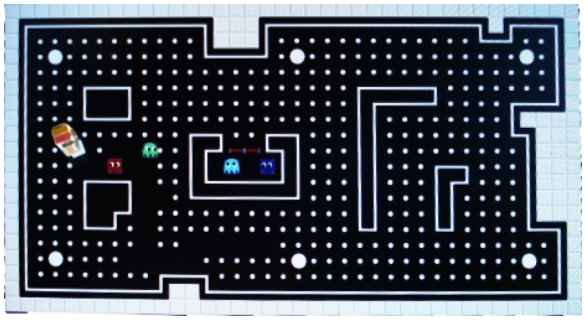


Figura 8: *Modificação do Jogo Pac-Man implementada*

A utilização da realidade mista possibilitou a fácil construção do ambiente virtual que, em conjunto com robôs reais promoveu maior realidade ao jogo. No Pac-Man, por exemplo, as pastilhas e pílulas especiais devem ser virtualmente representadas na tela pois elas podem desaparecer durante o jogo. No Battle City, o tiro disparado por um tanque é representado como um objeto virtual. Estes efeitos, que não podem ser reproduzidos em um ambiente real, só puderam

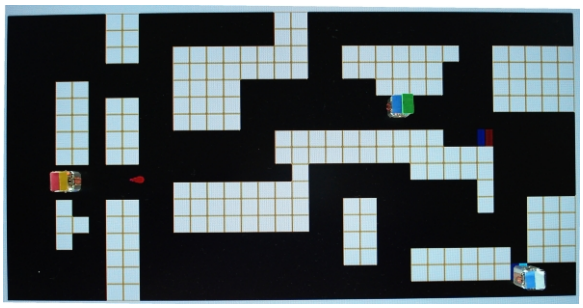


Figura 9: *Modificação do Jogo Battle City implementada*

ser representados pois utilizam um ambiente misto (que mistura o real com o virtual). Além disso os objetos reais são redesenhados na tela

Para mapear o cenário desses jogos, utiliza-se uma matriz de caracteres, onde cada caracter é representado na tela por um bitmap. Se ocorrer algum evento que altere algum dos Objetos Móveis, haverá então uma mudança na posição correspondente da matriz. Um exemplo de evento que altera os Objetos Móveis são as colisões entre o robô Pac-Man e as pastilhas. Outro exemplo é a colisão entre os tiros disparados por um dos tanques e a parede, que modificam o ambiente do jogo Battle City. Para analisar estes eventos de colisão recebe-se as posições dos Objetos Móveis e faz-se uma conversão de unidades para ter a posição real do objeto em relação a matriz.

Os eventos de colisão de Objetos Móveis entre si são feitos comparando as posições recebidas pela câmera: se o retângulo formado pelos pontos x e y iniciais e finais de dois ou mais elementos possuem pontos em comum então estes elementos colidiram. Uma colisão entre dois Objetos Móveis é pré-detectada a partir da informação do ângulo e módulo destes objetos. Isto ocorre pois se houver perigo de dois objetos reais estarem muito próximos ou se um objeto real estiver perto demais de uma parede, então este(s) objeto(s) deve(m) parar para evitar a colisão e qualquer tentativa de seguir na mesma direção será negada. O objeto poderá girar ou ir para trás. Esta informação é enviada diretamente pelo ambiente virtual que só libera um movimento na direção original quando não houver mais perigo de colisão.

Em ambos os jogos, a simulação e representação virtual de todos os objetos da realidade mista, seja ele real ou virtual, móvel ou estático, foram implementadas utilizando a biblioteca de jogos *Allegro* que possui funções que facilitaram o desenvolvimento.

6 Resultados Iniciais

Como a sub-liga PV é uma categoria nova, realizada pela primeira vez na Robocup 2007, ainda não existem resultados publicados por qualquer dos participantes desta competição. Partindo deste princípio, propõe-se metas a serem alcançadas nos três jogos: Futebol, Pacman e Battle City.

Em relação ao futebol, a estratégia e o controle propostos funcionaram como esperado, com os robôs cooperando entre si. Como a RoboCUp 2007 foi o primeiro campeonato, é possível aprimorar a estratégia para melhores resultados.

Com relação ao Pacman e ao Battle City pode-se afirmar que a estrutura proposta neste artigo está funcionando conforme previsto. A visão está robusta, identificando com perfeição todos os objetos e fornecendo informações precisas para a atualização do estado dos objetos simulados. O controle dos robôs é satisfatório, não prejudicando a jogabilidade, mas há espaço para melhorias nesse aspecto.

O ambiente usando futebol, Pac-Man e o Battle City foram apresentados na competição de demonstração da PV na Ro-

bocup 2007, em Atlanta, quando obteve o quinto lugar na classificação final.

7 Conclusões e Trabalhos Futuros

Neste artigo, primeiramente foi apresentada a arquitetura cliente-servidor da sub-liga PV da RoboCup, utilizada no futebol de robôs. Depois foi mostrado um resumo da arquitetura genérica desenvolvida para a sub-liga PV, que propõem a criação de aplicações originais para os mini-robôs. Esta arquitetura pode ser decomposta basicamente nos módulos de: Objetos Fixos, Objetos Móveis e Comunicação. Também foram ressaltadas as principais características da visão, planejamento e comunicação. E finalmente foram apresentadas adaptações dessa arquitetura para os jogos Pac-Man e Battle City.

Ênfase foi dada para a visão e o planejamento, demonstrando como foram implementados e as técnicas utilizadas. Na visão mostrou-se o uso do espaço de cores HSV para representar os pixels, como foi feita a classificação dos mesmos a partir de um conjunto de classes pré-definidos para o futebol de robôs e como os objetos foram localizados, a partir da identificação de manchas de classes de cores. O módulo de planejamento implementado para o futebol de robôs também foi demonstrado, ressaltando a máquina de estados criada para a equipe, os estados de cada agente, identificados por um estágio de percepção, e as técnicas de controle utilizadas. O Método de Decomposição de Células Aproximada juntamente com o Algoritmo de Dijkstra foram usados para a geração das trajetórias ótimas dos robôs no campo.

Existem vários planos de incrementar o Pac-man e Battle City, visando torná-los mais atrativos. Dentre as modificações desejadas, está a modificação do Battle City para possibilitar o jogo entre duas equipes remotas, a agregação de novos objetivos e também criação uma inteligência artificial para os agentes (tanques). Para o jogo Pac-Man pretende-se criar uma IA para possibilitar a cooperação entre os fantasmas explorando assim o controle dos robôs.

Referências

Amato, N. 2004. Randomized motion planning. Tech. rep., University of Padova.

Bruce, J., Balch, T., and Veloso, M. 2000. Fast and inexpensive color image segmentation for interactive robots.

Carvalho, F. G., Raposo, A. B., and Gattass, M. 2007. Uma interface híbrida para desktop integrando realidade virtual, realidade aumentada e 2d wimp. *IX Simposium on Virtual and Augmented Reality (SVR) 2007,Petrópolis, RJ, Brasil*, 152–161.

Cormen, T., Leiserson, C., Rivest, R., and Stein, C. 2001. *Introduction to Algorithms*. McGraw Hill/MIT Press.

Gonzalez, R., and Woods, R. 2001. *Image Processing*. Addison Wesley Pub.

Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., and Osawa, E. 1997. Robocup: The robot world cup initiave. In *The First International Conference on Autonomous Agent (Agents-97)*.

Latombe, J. 1991. *Robot Motion Planning*. Kluwer Academic Publishers.

Tamura, H., and Yamamoto, H. 1998. Vision and graphics in producing mixed reality worlds. *Computer Vision for Virtual Reality Based Human Communications, 1998. Proceedings., 1998 IEEE and ATR Workshop on*, 78 – 85.

Robot ARena: an Augmented Reality Platform for Game Development

Daniel Calife[1] João Luiz Bernardes Jr.[1] Romero Tori[1 2]

[1]Universidade de São Paulo, Depto. de Engenharia de Computação e Sistemas Digitais, Brasil

[2]Centro Universitário Senac, Brasil

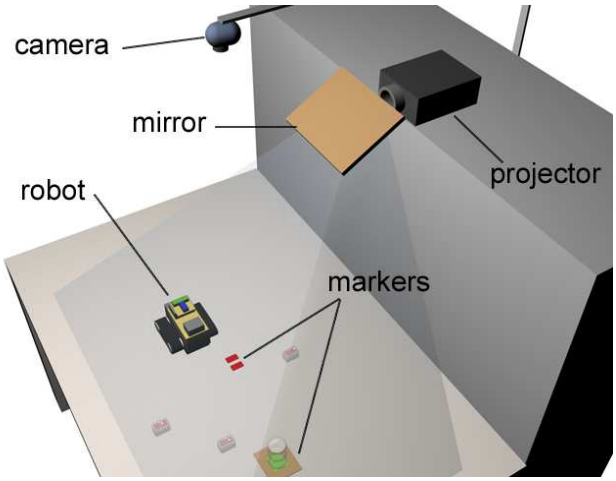


Figure 1: Robot ARena hardware setup.

Abstract

Augmented Reality is one of the new frontiers to be explored in electronic games. This paper presents Robot ARena, a platform for development and test of innovative games using Spatial Augmented Reality and a remotely controlled robot or tangible interfaces. Two game prototypes are implemented using this platform, and the results are discussed. One of these prototypes makes use of the real robot as an interaction element, while the other uses a tracked device as a tangible interface.

Keywords: augmented reality, games, robots, tracking.

Authors' contact:
{daniel.calife, joao.bernardes,
romero.tori}@poli.usp.br

1. Introduction

Although electronic game interfaces have evolved in time, enabling the creation of more graphically sophisticated environments and providing more elaborated controls, the way of interacting with games has not suffered any major change, mostly still using directional controls and push buttons, or mouse and keyboard based interaction. Augmented Reality (AR), combining 3D real and virtual environments interactively, breaks this paradigm and makes new ways of interacting and interfacing with games possible. It also allows a collaboration aspect for this environment and stimulates a more direct interaction between players, making it even possible to eliminate avatars or text-only chat interfaces within the game, extending its social aspect. Magerkurth et al. [2004] affirms that in traditional electronic games, computers and consoles, this type of interaction does not exist, due to its technology-centered, limited interactions between players. With AR, the game may be brought to the real world and people can have a more physical, personal and emotional interaction, as in traditional board or card games.

Aiming to analyze the new possibilities brought by the application of AR in games, Nilsen et al. [2004] characterize player experience in four aspects: physical, mental, social and emotional. Based on these aspects they compare the features of traditional and electronic games. No game uses only one of these aspects; the majority combines all of them. While electronic games show advantages in some aspects, traditional games are better in others. Computer games, for instance, shine in the application of Artificial Intelligence, attractive Computer Graphics, remote communication and the automatic calculation of game rules. Traditional games often have more natural interfaces and a richer social interaction between players. The great advantage of the development of games with AR is the possibility to mix the best features of both, making AR one of the important new frontiers to be explored in games.

Nowadays research projects in AR games can be classified in two broad categories, indoor and outdoor [Bernardes Jr. et al. 2005]. These classifications are based on the physical space demanded by the game: while outdoor games occupy large areas, indoor games usually require small, prepared spaces. This factor has

a great impact on the technologies and implementation of the games, especially for registration and interaction. Outdoor games mostly use physical metaphors of interaction, such as colliding with virtual objects to collect them, touching other players, running or walking in some ample physical space to move within the game. Indoor games also use the physical metaphor, but to a lesser extent, often in the form of Tangible User Interfaces (TUI) [Ishii and Ullmer 1997] and combining it with other interaction metaphors, such as player gesture or voice recognition.

This work presents an infrastructure, called “Robot ARena”, which can be used as a platform for development of innovative indoor AR games based on Spatial Augmented Reality (SAR) [Bimber and Raskar 2005] (which brings the visualization of the augmented content directly into the physical space) and a wireless controlled robot or tangible interfaces. The main goal of the Robot ARena project was to develop and test that infrastructure, exploring new ways of combining low-cost tracking, robot controlling and spatially AR techniques.

One of the main concerns during this research was to provide a basis to develop games that further explore the interaction between real and virtual environments, not only with real elements influencing virtual ones, but also the with the opposite happening, i.e. virtual elements affecting the real world. A wireless controlled robot is therefore adequate as a real "character" in the game, since it can both influence and be influenced by these virtual elements.

This infrastructure has as base the enJine, described in section 3.1, a game engine that provides a framework for the creation of virtual worlds and the necessary mechanics for the development of a game.

2. Related Work

Since the end of the last decade, several applications using AR as an interface have been described in the literature. More recently several projects combining AR and games have also been presented. In the field of robotics, in particular, computer vision and augmented reality have been used and combined even more numerously. This brief review will be limited to more recent projects, and those that are more closely related to the work described in this paper. The discussion begins with another extension of enJine's capabilities into the AR realm using JARToolKit and goes on about the DWARF framework for AR, the Augmented Coliseum where robots interact with virtual elements using a different control scheme and Kobito, which shares characteristics with the Robot ARena, such as the virtual elements affecting real objects.

2.1 EnJine + JARToolKit

Tsuda et al. [2007] present an integration of JARToolKit [Geiger et al. 2002] and enJine whose objective is the simplification of the development of low cost AR games, increasing the potential of enJine as a didactic tool and a testbed for new game technologies.

This integration with JARToolkit gives enJine the capacity to recognize commands given with the use of fiducial markers and a video camera. Besides reproducing the video stream obtained from the camera and tracking these markers. These characteristics, combined with enJine, facilitate the development of monitor-based AR games without the need for complex (and expensive) systems for tracking or stereoscopic visualization.

2.2 DWARF

DWARF (Distributed Wearable Augmented Reality Framework) [MacWilliams et al. 2003] is a framework based on distributed services (position tracking, three-dimensional rendering, input and output and task modeling) for Augmented Reality applications. Each service is a piece of software that can run on separated hardware components, with its own processor, memory, i/o devices and network access. The different services are dynamically connected with other services communicating their needs and abilities through the network. Distributed middleware is responsible for controlling the communications between these services.

This framework is used to develop complete Augmented Reality applications, projected to be flexible and guarantee to users a high degree of freedom. These characteristics are attained because the framework's services can be distributed over independent machines, that can be added or removed from the environment without causing a great impact on the whole system.

2.3 Augmented Coliseum

Augmented Coliseum [Kojima et al. 2006] is an application similar to the game prototype described here. It is a game in which Augmented Reality is used to enrich a competition between real robots with effects such as rays and explosions in a virtual environment. Robots can also collide with obstacles in the virtual environment that can block their movement. Unlike the work described in this paper, however, there is no need for visual tracking. The position of the robots is always known because they actively follow a certain pattern projected on them along with the virtual environment they are in. Each robot has 5 light sensors in known positions that detect variations in this light pattern and can identify where it is moving to and then follow it.

2.4 Kobito

An interesting project that makes use of augmented reality and virtual and real objects interacting "physically" is the Kobito project [Aoki et al. 2005]. In this application, a real object, a tea caddy on a table set for a tea break, is moved around by "invisible brownies", virtual agents that can only be seen through a special display, the "Kobito Window", which is nothing but an AR interface for visualization. The caddy is actually moved around by a magnet under the table, which, in turn, is manipulated by the SPIDAR system. The virtual agents not only manage to push the real caddy around, but can also be pushed by it (in fact, that is the only way to interact with the brownies, through a real object) if a user manipulates it. The caddy even offers force feedback, representing resistance from the brownies. The system makes use of physical simulation, but unlike its classical applications, where all simulated elements are virtual, there is a real element that must be tracked precisely (this task is done with a camera) and manipulated (with the SPIDAR system). Unlike the manipulation system used in the Kobito project, the robot used so far in the Robot ARena can only turn and move back and forward (it cannot be pushed sideways, for instance), so the degree of physical interaction with virtual objects will be necessarily limited and must be designed to hide this problem as much as possible from the user.

3. Used Tools

This section presents the tools used to develop both the logical and physical components of Robot ARena.

3.1 EnJine

EnJine is an open-source didactic game engine [EnJine 2007; Nakamura et al. 2006] that uses Java and the Java3D library. It is developed and supported by the Interactive Technologies Laboratory (Interlab) at University of São Paulo, and available as free software under the GPL license.

EnJine's main purpose is to serve as a teaching support tool in computer science courses, especially for computer graphics [Tori et al. 2006]. It is therefore didactic and relatively simple to learn and use, and has an architecture which facilitates good software engineering practices. Another purpose for enJine is to serve as a framework for implementing and testing new technologies, mainly applied to games.

EnJine provides a set of services for game development, as well as a structure for the creation of these games. Its main features are:

- A set of Java classes to allow flexible representation of game elements such as characters, objects and scenarios;
- Basic infrastructure for a game application, including the capability for complex interactions between game objects through a message exchange mechanism;
- 3D graphics rendering through the Java 3D API;
- Multi-stage collision detection and ray-casting collision detection;
- Skin-and-Bones animation;
- Support for basic sound output and user input.

EnJine's architecture is divided in three abstraction layers. The lowest layer represents the external libraries used to communicate with the operating system and devices, and some other services such as Java3d's rendering. The middle layer contains enJine's main modules, which implement the games' services, such as video rendering, sound output, collision detection and others. The top layer implements enJine's framework, which is a set of classes intended to provide a more complete and easier solution for the implementation of specific types of games, as opposed to enJine's more general nature. The framework includes concrete implementations of many of enJine's classes suited to specific uses. For instance, the SinglePlayerGame class implements the base code to aggregate enJine service providers needed for a computer game that runs in a single machine. Figure 2 shows these abstraction layers.

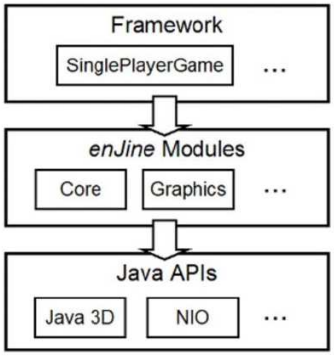


Figure 2: EnJine's architecture, from Nakamura et al. [2006].

The software infrastructure developed in this work extends enJine's middle layer, which is composed by the packages presented in figure 3.

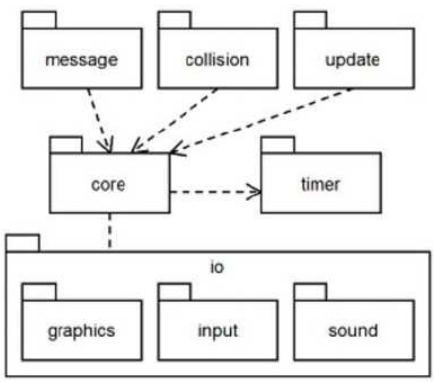


Figure 3: EnJine's main packages, from Nakamura et al. [2006].

3.2 OpenCV

OpenCV [2007] (Open Source Computer Vision Library) is a free library of C/C++ functions mostly for computer vision and image processing, with a focus on real-time applications, which makes it interesting for Augmented Reality.

As two of the main concerns of Augmented Reality are registration and tracking, and one approach to obtain them is through computer vision, OpenCV is a robust and flexible tool to be used in such applications.

3.3 Lego Mindstorms

Lego Mindstorms [2007], or Lego Robotics Invention System (RIS), is a set of blocks, motors, sensors and other components, such as gears and axles, created and marketed by the Lego group. It can be used to build many types of robots and other automated or interactive systems.

The main component of Lego Mindstorms, which controls input (sensors) and output (motors), is called the RCX block. The RCX has a Renesas H8/300 microcontroller that interprets the programs loaded in its RAM through an interface that uses infrared to send and to receive data.

3.4 JavaComm

The Java Communications API [2007] (JavaComm) is a Java library that aids the development of platform-independent applications for communication. This library provides application access to RS-232 and IEEE-1284 hardware.

4. Robot ARena

Robot ARena is a software and hardware platform for the development of AR games. It has a robot as its main interaction element, but is not limited to using only that. The wireless controlled robot was chosen because of its capability not only to affect but also to be affected by virtual elements. It can collect some virtual element passing over it, for instance, or collide with and be hindered by virtual barriers or be pushed by virtual characters.

For the visualization of the augmented environment, Robot ARena uses the concept of projection-based spatial displays, presented in [Bimber and Raskar 2005], freeing users from uncomfortable devices, such as Head-Mounted displays and, at the same time, enabling the visualization of the virtual content directly on the real world.

Robot ARena is implemented in a game structure as an abstraction layer above enJine, creating new possibilities for visualization and game code. Figure 4 shows a common structure of a game.

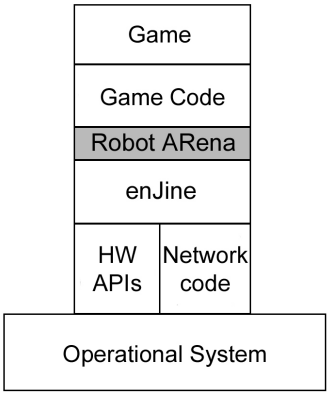


Figure 4: Robot ARena in Game Structure, adapted from [Lewis et al. 2002].

Although the main interaction element of the platform is a robot, it is not necessary that all games, or applications developed, use it. The platform provides two tracked devices, described in section 5.1.2, which can be attached to any desired object, or be used by themselves as tangible elements of interaction, as shown in section 6.2.

The platform enables the insertion of real elements, aside from the main tracked object, that can also influence virtual elements. These additional real elements, however, are not tracked and therefore must be static and have their position determined during the system's initialization, using color markers positioned in the arena.

4.1 Hardware

Robot ARena's hardware is composed of a table, a camera, a projector, a mirror, a robot and two tracked devices, all of them connected through a computer, except for the tracked devices (detected through computer vision), in the setup shown in figure 1.

4.2 Software Architecture

Robot ARena's software is divided in three subsystems, some of which directly access its hardware (treated as a black box) sending and receiving signals. These subsystems, and their relationship with each other, are shown in figure 5, and discussed in the next sections.

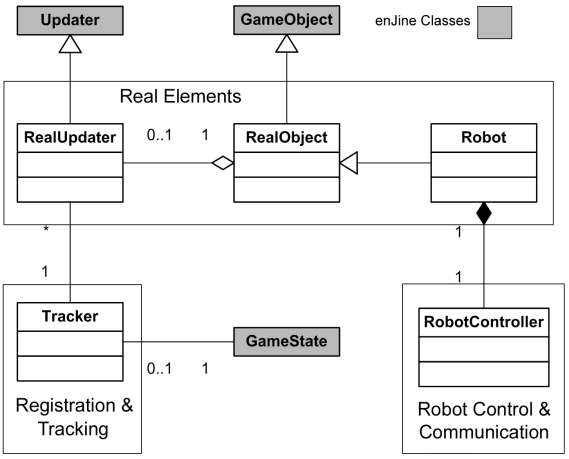


Figure 5: Robot ARena software architecture.

4.2.1 Registration and Tracking

This subsystem is a software component that provides an interface, which has methods to do the registration of the virtual and real environments, identify real elements that affect the game and track the robot, (or other object) updating its position and orientation.

4.2.2 Real Elements

A set of classes, which, through inheritance from enJine's GameObject, specify the features of game elements that have a physical equivalent. A more specialized class derived from RealObject is Robot.

The real position and orientation of these elements, including the robot, is obtained from the Registration and Tracking subsystem.

4.3.3 Robot Control and Communication

This class provides an interface for control and communication of the real robot, allowing the player, as well as the virtual elements that can interact with it, to send movement commands to the robot.

5. Robot Arena Implementation

This section describes the implementation of Robot ARena's hardware and software.

5.1 Hardware Implementation

5.1.1 Robot

The robot is assembled upon the RCX block, controlling its input sensors and motors. Its tracked vehicle design, with differential steering, results in a neutral turn, i.e. it is possible to change its orientation without changing its central position. This is important to simplify the treatment of collisions. Figure 6 shows the robot.

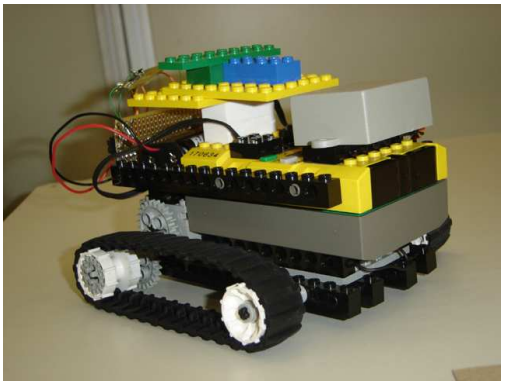


Figure 6: Robot used in the infrastructure.

The communication between computer and robot is achieved through a RF transmitter module in the computer and a receiver module in the robot. Each module has a micro controlled 2.4GHz transceiver. The diagram in figure 7 shows an abstraction of this communication system.

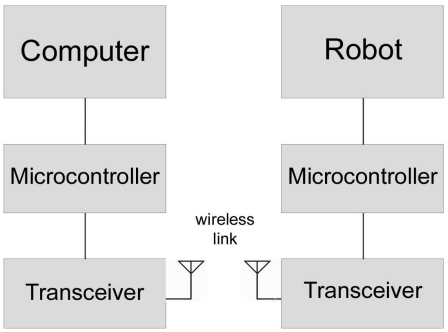


Figure 7: Robot communication system.

The data received by the communication system are 3-bit words. Each bit defines the logical value of one of the three input sensors in the robot. Two of these sensors control the direction of each motor, one for each track (0 makes it move backwards and 1 forward), while the other sensor indicates if both motors are on (1) or off (0). This control scheme allows the basic movements of the robot: moving forward or back, turning left or right and stopping [Calife et al. 2006].

5.1.2 Tracked Devices

Robot ARena has the capability to track two different types of devices. One of them is actually no more than a color marker, formed by two Lego blocks, while the other is a device with two infrared LEDs.

The color marker is specially created for robot tracking, its two Lego blocks, one blue and one green in order to differentiate one from the other, draw a “T” marker, that can be assembled on top of the robot. This marker is shown in figure 8.

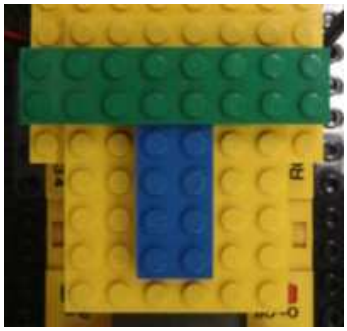


Figure 8: Robot's tracked marker.

The infrared tracked device is a small box with two LEDs, used for general purposes, which can be easily manipulated by users and attached or associated with some other object. Its two LEDs work much like the two colored blocks in the robot marker. To differentiate the two LEDs a digital circuit lights one of them with more intensity than the other. A representation of this device can be seen in figure 9.

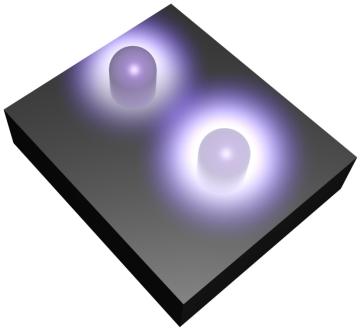


Figure 9: Representation of Infrared tracked device.

5.1.3 Camera, Projector and Mirror

Above the table, there is a camera, a common webcam, aligned so that its viewing direction is approximately perpendicular to the table plane.

A common projector is used, connected to the computer video card and projects exactly the same content presented in the monitor display. To direct this projection, a mirror is used. Usually, this mirror would be positioned at an angle of 45º in relation to the projector, resulting in a 90º reflection of the projection. In this configuration, however, the mirror would be located precisely at the projection center and would obstruct the camera's vision completely. To solve this problem, the mirror is positioned at an angle of 30º, aiming the reflected projection at an angle of 120º, as figure 10 shows. This configuration adds some distortion to the projected images, which is corrected through manual calibration of projector and video card settings based on the projected and known images.

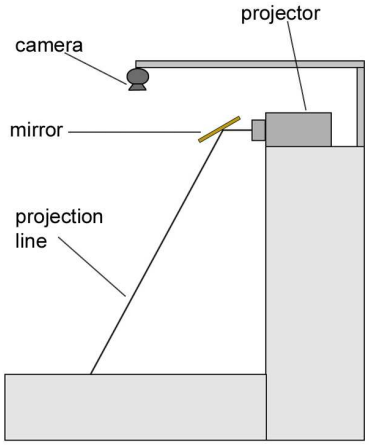


Figure 10: Camera, projector and mirror setup.

5.2 Software Implementation

5.2.1 Registration and Tracking

The Registration and Tracking subsystem is implemented in C++, using the OpenCV library. For the integration of this component with enJine (Java) the JNI (Java Native Interface) was used, which defines an interface to create a DLL, allowing its methods (written in C++) to be called within a Java class. To facilitate this process, a simple implementation of this subsystem was chosen, implemented in only one class and using only native types in both languages as return types for its methods.

The Tracker class operates in two stages: initialization and tracking. In the first stage, usually implemented inside the setup stage of a game, the Tracker object initializes the camera, makes the initial registration of the scene and finds the additional elements located on the Robot ARena (init(), findElements()). At the initialization stage, it is also possible to determine the position of the additional elements found (getNextElement(),getElementX(),getElementY()).

The second stage is responsible for tracking the devices described in 5.1.2. A parallel thread executes the getPosition() method continuously, updating the position and orientation of the tracked object (usually the robot). In each update cycle, the RealUpdater class gets the last known position from the tracked object and updates its logical state. Figure 11 shows this class.

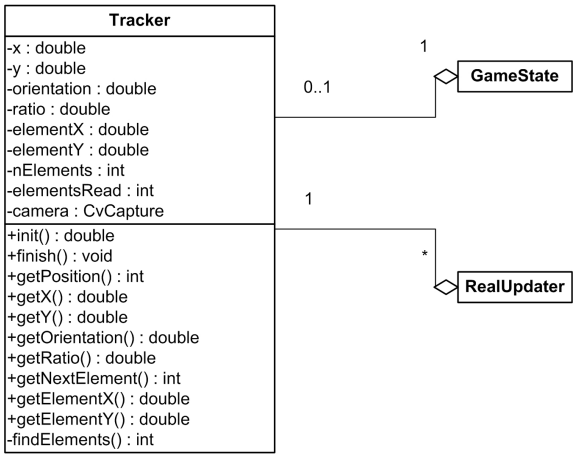


Figure 11: Register and Tracking diagram.

Registration and Tracking Approach

The approach used to accomplish registration and tracking in Robot ARena is based color segmentation, which is why colored markers are used. The only exception is the infrared device, but the brightness emitted by its LEDs, seen only by the camera, posses a strong white coloration and is of easy segmentation through image processing.

At the center of the ARena is fixed a red marker, used to calibrate the registration of the scene, serving as a reference point for the origin of real and virtual environment coordinate systems. Because its physical properties are known, it can provide a scale factor to the transformation between the real and virtual coordinate systems. Other color (green) markers can be freely located in the ARena, or attached to a real object (not tracked), representing the additional elements recognized in the initialization stage.

Both tracked devices must have two different points of reference so it is possible to determine an origin and an orientation for the tracked object.

More details of this implementation can be seen in [Calife et al. 2007].

5.2.2 Real Elements

Real Elements are classes that inherit from two of the main classes of enJine's Core package: GameObject and Updater. The RealObject class inherits from GameObject and basically implements the attributes and methods necessary to define and manipulate a real object, whether tracked or not. Usually, RealObject represents additional real elements located in the Robot ARena and found by the Registration and Tracking subsystem. The Robot class inherits from RealObject and aggregates a RobotController object, described in section 5.2.3. It implements the methods to control the robot's movements.

The RealUpdater class inherits from enJine's Updater class, which has the responsibility of updating the logical state of game objects. RealUpdater has as an attribute a Tracker object, described in section 5.2.1, which updates the tracked position and orientation of the RealObject or Robot through the updater() method. These classes can be seen in figure 12.

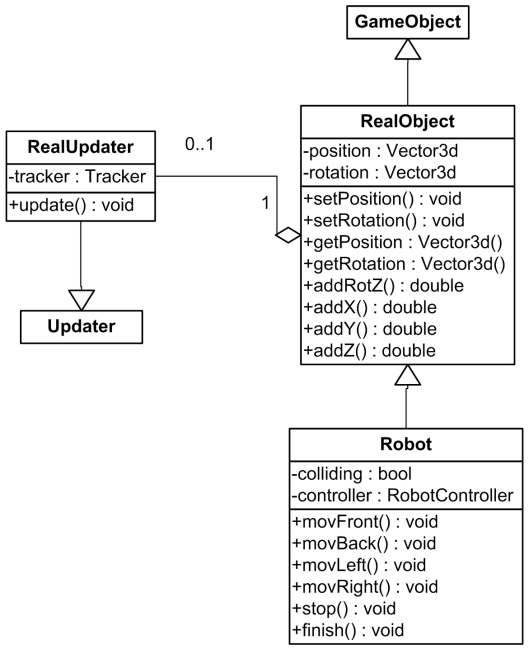


Figure 12: Real Elements classes.

5.2.3 Robot Control and Communication

The Robot Communication and Control subsystem is implemented in the RobotController class, which is part of the Robot class, described in section 5.2.2. This class is implemented with the aid of the JavaComm library, which initiates and controls the communication through the serial port (RS-232). The methods that indicate the movements of the robot (front(), back(), left(), right() and stop()) send the 3-bit words representing movement to the transmitter module hardware, described in section 5.1.1. Figure 13 shows this class.

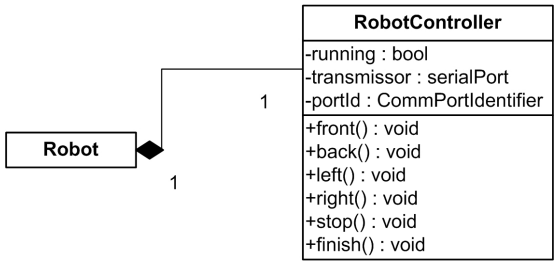


Figure 13: Robot Control and Communication Diagram.

6. Prototypes

As an implementation test for the platform, two game prototypes have been developed, both acting as a proof of concept for Robot ARena. The first one, FootBot ARena, implements all features of the platform, with the user controlling the robot. The second game, TanSpace, is implemented to test the infrared tracker device, used as a Tangible Interface.

6.1 FootBot ARena

FootBot ARena is a simple game that, despite its simple mechanics, offers the necessary environment for the interaction of the robot with the virtual elements present in the game.

The basic mechanics of the game is to move the robot so that it collects projected virtual batteries by moving over them. There are obstacles, virtual metal barrels positioned in the game area, that block the robot's movement. If the robot collides with one of these virtual barrels, it will not be able to move in that direction, simulating a real collision, being forced to move backwards and go around the obstacle. Any number of virtual obstacles can be placed in the game area. These obstacles are represented in the real world by green markers, which are the additional virtual elements, and their positions are recognized in the initialization stage of tracking.

Besides this basic mechanics, three game modes were created: unlimited, time attack and timed.

In the first mode, unlimited, the game simply begins, placing the virtual batteries and barrels in the augmented environment, and the robot's movements are turned on, allowing the player to control it. In this mode, the player can take as long as he wishes to collect all the batteries.

In time attack, a timer is added to the game and stops as soon as all batteries are collected, so players can challenge one another to see who records the best time.

In the last mode, timed, the timer counts down from a preset limit, instead of marking elapsed time, so that players have a limited time to collect all the batteries. If the clock reaches zero, the robot stops responding to the player's inputs and a counter shows how many batteries were collected.

While this describes the game's prototype version, future versions promise to be more complex, as discussed in section 7. Figure 14 shows the prototype running, with the robot, four batteries and one metal barrel, as an obstacle.

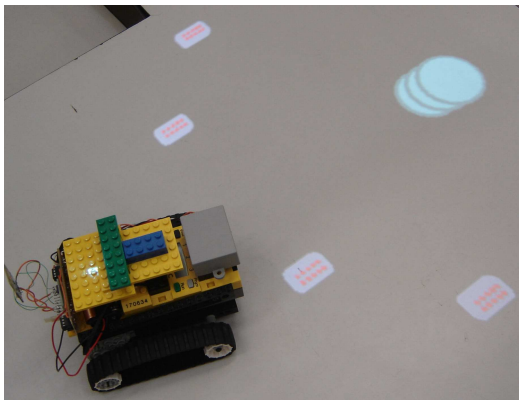


Figure 14: FootBot ARena prototype running.

6.2 TanSpace

The main purpose of this game is to explore the possibilities of tangible user interfaces, with the infrared tracked device being manipulated directly by the user.

The game has interstellar space as its scenario, with the player piloting a ship that must destroy enemy vessels. To do that, the player must control an aiming sight, placing it over the enemy ship and lining it up with a second sight that moves with the enemy.

The TUI device is capable of determining the position of this sight, allowing the player to freely place the device over the enemy ship and to follow it. However, as the main purpose of this type of interface is to free users from keyboard and mouse, there was a problem to send the command to shoot enemy ship, without pushing any button. A creative way was implemented to solve this problem. Each enemy ship has an indication, in the form of red arrowheads, that define the best position for the shot, and the sight the player controls also has two arrowheads, which must be lined up with the ones on the ship. Only when the sight is located on the enemy ship and both arrows are lined up, the player's ship shoots, destroying the enemy. In this way the game makes use of the characteristics of the developed infrared tracked device, the determination of the position and the orientation, and frees the players from the keyboard and mouse, making a much simpler and more intuitive interaction possible. Figure 15 shows a screenshot of the game.

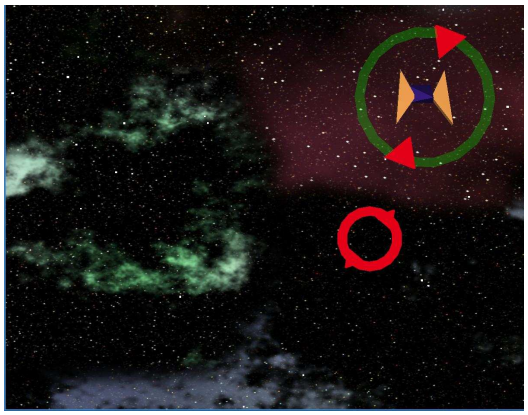


Figure 15: TanSpace prototype's screenshot.

6.3 Results

The results achieved with the development of these two games prototypes using the Robot ARena framework are satisfactory and promising, fulfilling the objectives presented for this platform.

The main purpose, the development of a platform that allows the development of games with interaction in both directions between real and virtual elements, was explored and attained through the combination of all techniques presented.

EnJine has proven to be an adequate choice as the system's core, facilitating the creation of the virtual environment, the integration between different modules of the system and for developing game mechanics.

Some performance measurements [Calife et al. 2007], application frame rates, were obtained, and these also demonstrated to be satisfactory. For these measurements, a PC with the following configurations was used: Athlon XP 2200+ processor, 512MB of RAM memory and a 3D accelerator video card with 128MB of memory. The tracking algorithm (position and orientation) running isolated, without graphical rendering, reaches the average frame rate of 52,33 frames per second. Including all the 3D rendering process, collision detection, interaction and integration of the Robot ARena's modules, all controlled by enJine, an average frame rate of 16.78 frames per second was measured during the prototype game.

The FootBot ARena prototype uses all capabilities of the platform, and makes use of all software developed, which facilitated the interaction between players and robot, with a ready-to-use communication system. The robot interacts with the virtual world, which makes it look as though they coexist, thanks to the tracking system. The use of a projection display allows a much more comfortable visualization of the augmented environment, making the presence of many spectators possible during the player's interaction with the game. For these reasons a top view with a fixed isometric projection was chosen for the augmented environment visualization. With the use of additional elements it is possible to change the game scenario as desires, including the possibility of increasing or decreasing the difficulty level.

A problem that occurs during the projection of the augmented environment is the interference of the luminosity level on the tracking system. Because this system is based on color segmentation, and especially on saturation, the ARena must have a good illumination, which affects the perceived brightness of the projected elements.

This problem can be solved with the use of the infrared LED tracking system. The tracking algorithm is very similar, with the advantage of the infrared light not suffering direct influence from the surrounding illumination and the projection.

While TanSpace does not use the robot features that the platform provides, it implements a tangible interface and shows that the platform can be used to develop many types of games and applications.

TanSpace's prototype could be developed in only two days, by one person, thanks to enJine and the Robot ARena framework. For FootBot ARena this measure of development time does not exist, because it was implemented almost at the same time as Robot ARena.

7. Conclusions and Future Work

Robot ARena is a test bed for developing games and applications involving robots and augmented reality, from educational tools to innovative augmented reality games. New techniques, in areas such as tracking, artificial intelligence, interaction and projector-based augmented reality, can also be researched and tested with the Robot ARena infrastructure.

An interesting possibility that Robot ARena brings is the capacity to have applications involving remote users, such as a game over the Internet, where each player controls its proper real robot in a match where the opponent's robot is presented as a virtual robot, or as a robot video avatar.

The FootBot ARena prototype presented here is only a small part of the complete game that is under development. It will be a soccer game with robots, and will explore yet more interaction possibilities between virtual and real elements, implementing, for instance, any combinations of disputes between real or virtual robots. The ball will be virtual, adding new tasks for physical simulation.

Another feature for future versions of FootBot ARena is to use SAR to change the visual appearance of the real elements in the game, the robot, for instance, changing its color to represent a certain team

of which the player is part, or adding some special effects, such as blinking in red when it is close to exploding.

It is possible to provide a better representation of the augmented environment with more depth cues, projecting its content according to the player's viewpoint, instead of using an isometric projection, but this approach reduces the player's freedom of movement, unless his head position is also somehow tracked. This view-dependant projection would also reduce the possibility of a comfortable view for spectators (they could only stand close to the player for the best view).

Assuming that the player's viewpoint is tracked, it is possible to use stereoscopic projections, providing more realism to the augmented environment, a feature that is currently being researched.

Acknowledgements

The authors would like to thank all Interlab students, researchers and professors. And, also, thank CNPq and ItauCultural´s Rumos Arte Cibernética for Daniel Calife's master research financial support.

References

AOKI, T., ICHIKAWA, H., ASANO, K., MITAKE, H., IIO, Y., AYUKAWA, R., KURIYAMA, T., KAWA, T., MATSUMURA, I., MATSUSHITA, T., TOYAMA, T., HASEGAWA, S. AND SATO, M. 2005. KOBITO - VIRTUAL BROWNIE - VIRTUAL CREATURES INTERACT WITH REAL OBJECTS AND REAL PEOPLE. *SIGGRAPH'2005 EMERGING TECHNOLOGIES*.

BERNARDES JR., J.L., DIAS, J.S. AND TORI, R. 2005. EXPLORING MIXED REALITY USER INTERFACES FOR ELECTRONIC GAMES. *IN: ANAIS DO IV WORKSHOP BRASILEIRO DE JOGOS E ENTRETENIMENTO DIGITAL (WJOGOS'2005)*, SÃO PAULO, P. 353-358.

BIMBER, O. AND RASKAR, R. 2005. *SPATIAL AUGMENTED REALITY: MERGING REAL AND VIRTUAL WORLDS*. WELLESLEY: A K PETERS.

CALIFE, D., TOMOYOSE, A., SPINOLA, D., BERNARDES JR., J.L. AND TORI, R. 2007. ROBOT ARENA: INFRAESTRUCTURE FOR APPLICATIONS INVOLVING SPATIAL AUGMENTED REALITY AND ROBOTS. *IN: PROCEEDINGS OF THE IX SYMPOSIUM ON VIRTUAL AND AUGMENTED REALITY (SVR'2007)*, PETRÓPOLIS.

CALIFE, D., TOMOYOSE, A., SPINOLA, D. AND TORI, R. 2006. CONTROLE E RASTREAMENTO DE UM ROBÔ REAL PARA UM AMBIENTE DE REALIDADE MISTURADA. *IN: ANAIS DO II WORKSHOP DE APLICAÇÕES DE REALIDADE VIRTUAL (WARV'2006)*, RECIFE.

ENJINE: ENGINE FOR GAMES IN JAVA HOMEPAGE. HTTPS://ENJINE.DEV.JAVA.NET// [ACCESSED 12 OCTOBER 2007].

GEIGER, C., REIMANN, C., STICKLEIN, J. AND PAELKEE, V. 2002. JARTOOLKIT – A JAVA BINDING FOR ARTOOLKIT. *IN: PROCEEDINGS OF THE FIRST IEEE INTERNATIONAL WORKSHOP ON AUGMENTED REALITY TOOLKIT*.

ISHII, H. AND ULLMER, B. 1997. TANGIBLE BITS: TOWARDS SEAMLESS INTERFACES BETWEEN PEOPLE, BITS AND ATOMS. *IN: PROCEEDINGS OF THE SIGCHI CONFERENCE ON HUMAN FACTORS IN COMPUTING SYSTEMS*, ATLANTA, P. 234-241.

JAVA COMMUNICATIONS API HOMEPAGE. HTTP://JAVA.SUN.COM/PRODUCTS/JAVACOMM/ [ACCESSED 20 AUGUST 2007].

KOJIMA, M., SUGIMOTO, M., NAKAMURA, A., TOMITA, M., NII, H. AND INAMI, M. 2006. AUGMENTED COLISEUM: AN AUGMENTED GAME ENVIRONMENT WITH SMALL VEHICLES. *IN: FIRST IEEE INTERNATIONAL WORKSHOP ON HORIZONTAL INTERACTIVE HUMAN-COMPUTER SYSTEMS*.

LEGO MINDSTORMS HOMEPAGE. HTTP://MINDSTORMS.LEGO.COM/ [ACCESSED 20 AUGUST 2007].

LEWIS, M. AND JACOBSON, F. 2002. GAME ENGINES IN SCIENTIFIC RESEARCH. *COMMUNICATIONS OF THE ACM*, V.45, 1, P. 27-31, JAN.

MACWILLIAMS, A., SANDOR, C., WAGNER, M., BAUER, M., KLINKER, G. AND BRUEGGE, B. 2003. HERDING SHEEP: LIVE SYSTEM DEVELOPMENT FOR DISTRIBUTED AUGMENTED REALITY. *IN: PROCEEDINGS OF THE SECOND IEEE AND ACM INTERNATIONAL SYMPOSIUM ON MIXED AND AUGMENTED REALITY (ISMAR'2003)*, TOKYO, P. 123-132.

MAGERKUTH, C., ENGELKE, T. AND MEMISOGLU, M. 2004. AUGMENTING THE VIRTUAL DOMAIN WITH PHYSICAL AND SOCIAL ELEMENTS: TOWARDS A PARADIGM SHIFT IN COMPUTER ENTERTAINMENT TECHNOLOGY. *IN: PROCEEDINGS OF THE 2004 ACM SIGCHI INTERNATIONAL CONFERENCE ON ADVANCES IN COMPUTER ENTERTAINMENT TECHNOLOGY*, SINGAPORE, P. 163-172.

NAKAMURA, R., BERNARDES JR., J.L. AND TORI, R. 2006. USING A DIDACTIC GAME ENGINE TO TEACH COMPUTER SCIENCE. *IN: DIGITAL PROCEEDINGS OF THE V BRAZILIAN SYMPOSIUM ON COMPUTER GAMES AND DIGITAL ENTERTAINMENT, SBGAMES'2006 TUTORIALS*, RECIFE.

NILSEN, T., LINTON, S. AND LOOSER, J. 2004. MOTIVATIONS FOR AUGMENTED REALITY GAMING. *IN: PROCEEDINGS OF THE NEW ZEALAND GAME DEVELOPERS CONFERENCE (FUSE'04)*, DUNEDIN, P. 86-93.

OPEN SOURCE COMPUTER VISION LIBRARY HOMEPAGE. HTTP://WWW.INTEL.COM/TECHNOLOGY/COMPUTING/OPENCV/ [ACCESSED 20 AUGUST 2007].

TORI, R., BERNARDES JR., J.L. AND NAKAMURA, R. 2006. TEACHING INTRODUCTORY COMPUTER GRAPHICS USING JAVA 3D, GAMES AND CUSTOMIZED SOFTWARE: A BRAZILIAN EXPERIENCE. *INTERNATIONAL CONFERENCE ON COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES, ACM SIGGRAPH'2006 EDUCATORS PROGRAM*, BOSTON.

TSUDA, F., HOKAMA, P.M., RODRIGUES, T.M. AND BERNARDES JR., J.L. 2007. INTEGRATION OF JARTOOLKIT AND ENJINE: EXTENDING WITH AR THE POTENTIAL USE OF A DIDACTIC GAME ENGINE. *IN: PROCEEDINGS OF THE IX SYMPOSIUM ON VIRTUAL AND AUGMENTED REALITY (SVR'2007)*, PETRÓPOLIS.

Integrating the Wii controller with enJine: 3D interfaces extending the frontiers of a didactic game engine

João Bernardes Ricardo Nakamura Daniel Calife Daniel Tokunaga Romero Tori

Universidade de São Paulo, Dept. de Engenharia de Computação e Sistemas Digitais, Brazil

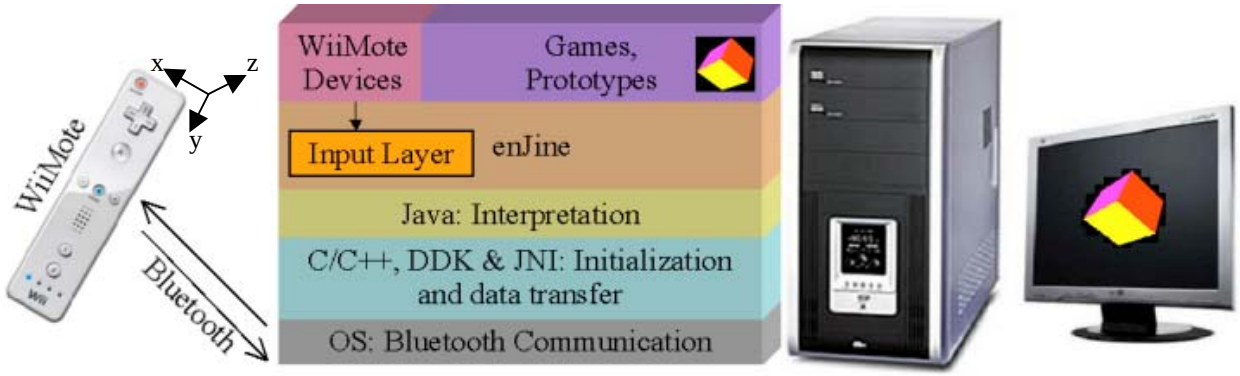


Figure 1. Integration of the WiiMote and enJine

Abstract

The goal of the work described here is to integrate a 3D device, the Wii Controller, and enJine, a didactic engine, motivated by the growing use of 3D interfaces. The paper discusses how this increases enJine's didactical and technological potential and details the adopted solution as a layered architecture. Two interaction styles were tested. Test results show a variety of data about the controller, that this solution works as desired and that using it to modify a game to use the WiiMote as its input device is a simple task.

Keywords: Tangible User Interface, Gesture-based Interface, Game Engine, Didactic Tool

Authors' contact:
{joao.bernardes, ricardo.nakamura,
daniel.calife, daniel.tokunaga,
romero.tori}@poli.usp.br

1. Introduction

This paper describes a work whose main goal is to integrate a 3D user interface device, the Wii Controller (also called Wii Remote or WiiMote), and enJine, a didactic 3D game engine. The advantages of using a game engine, and therefore games and game programming, as a didactic and motivational tool for students have been discussed by several authors. A similar discussion has been presented before including the use of enJine [Tori et al. 2006] and focused in teaching computer science or engineering. But what should be the advantages of adding a 3D input device in such a setup?

Games for non-portable platforms have, for several years now, been in their large majority 3D. Even for portable platforms a tendency towards 3D games exists, as shown, for instance, by the 3D engine for cellphones called M3GE [Gomes and Pamplona 2005]. This tendency is still slow only due to hardware cost. Using 3D game programming when teaching is not only useful to work with spatial concepts, necessary to the target students and some courses, such as Computer Graphics, but also helps in student motivation, since they tend to perceive these games as the "best" ones and their creation as a worthy challenge. Conventional input devices, however, such as joysticks, mouse and keyboard, were designed primarily for 2D applications and their use in 3D requires more sophisticate techniques and training. While this is usually not an issue for those already trained (such as avid gamers or users of 3D modeling or CAD applications), it is a barrier for users new to gaming. Non-conventional, 3D input devices and user-interface techniques address this issue by providing a more natural (and arguably fun) way of interfacing with 3D applications, and their use is already a strong tendency in electronic games, as demonstrated by the success of Nintendo's WiiMote [Nintendo 2007].

Wii is the last generation console from Nintendo, competing with Microsoft's XBox 360 and Sony's Playstation 3 (PS3). Wii has admittedly less processing and graphics power than its competitors and, despite that, has met unexpected success in the market. In the US alone it sold 600 thousand units up to 8 days after its launch, and 4 million units in 6 weeks [Dobson 2006]. The console was soon missing in the market and being sold by values overpriced by up to 40%, approximately. These numbers are far superior to those attained by Wii's competitors. The XBox 360, for instance, sold approximately 3 million units worldwide up to 90 days after launch [Carless 2005]. The natural and fun 3D interface embodied in the WiiMote is considered the main reason for this performance despite its otherwise inferior technical specification (Wii's lower price is also a factor). Several users have even bought both a Wii and a XBox 360, a pair

nicknamed "Wii60", to have both the 3D interaction and high-end graphics to play with. It is interesting to notice that Interlab had already been researching and publishing about new 3D interfaces for games in enJine for a while before the excitement over WiiMote. Soon after its appearance, Sony also announced motion-sensing capabilities for PS3's controller had been in research for over 2 years and would be present at launch. All this indicates that 3D and more natural interfaces are indeed a strong tendency in games (and hopefully in other applications as well).

This tendency is one of the reasons to integrate the WiiMote and enJine (as well as for other enJine extensions discussed in section 3). As a didactic tool for several computer science or engineering and even game programming courses, enJine must always strive to be up-to-date with the evolution of interface and game technologies. This not only influences student motivation, but also offers a broader range of options for each course. User Interface courses at both undergraduate and graduate levels, for instance, are often limited to 2D devices and techniques, but with the use of enJine and the WiiMote they could cover some 3D techniques as well, such as tangible interfaces. At a lower level of abstraction, even signal processing (to treat the raw data obtained from the WiiMote) could be explored didactically.

Another motivator for this integration is the fact that enJine's simplicity (one of the main requirements for its goal as a didactic tool) and flexibility, especially in its abstract input layer, reduce the complexity of such work. Finally, enJine has secondary goal. While its main objective is to be an effective teaching tool, its simplicity and flexibility have lead to a new goal: to serve as a test bed for new technologies in games. Lately it has been used primarily to explore Augmented Reality (AR) - a form of interface that combines real images, seen directly or captured in video, and virtual elements in a scene interactively and in real time with 3D registration between real and virtual elements [Azuma 1997] - and other new interfaces in games.

While several other 3D devices, such as 3D mice (magnetic or gyroscopic), could have been chosen as an input device for enJine, the choice of the WiiMote was not merely driven by "fashion". The main factors for this choice were the low cost and especially the availability of this controller, which, unlike other 3D devices, is widely accessible to the general public. Before a more detailed discussion of WiiMote, it is important to notice now that, aside from its use as a regular wireless game controller, it provides tracker-like data (what it actually measures is accelerations in 3 axes; it can also be used as a pointing device thanks to an infrared system), which can be used to implement both tangible and simple gesture-based interfaces.

Tangible User Interfaces (TUI) are defined as Computer-Human Interfaces that use interactions with the physical world as metaphors for information manipulation in the virtual world [Ishii and Ulmer 1997]. In these interfaces, a physical and a virtual object are associated so that changes in one (usually manipulation of the physical object) should affect the other. This association of the WiiMote with virtual objects in games is quite clear in several cases, such as when using the controller to represent pointing a gun, swinging a bat, a tennis racket or a sword. There are even kits of simple plastic devices that can be securely attached to the Wii to make it look like these objects. Often, however, there is not such a direct correspondence between the manipulation of the WiiMote and the in-game object and the gestures made by the user are what actually serve as a game input.

Considering that, for gestures, the WiiMote is nothing but a hand tracking device (both hands can be tracked if using the Nunchuk, an accessory that can be attached to the WiiMote), it only allows the use of pointing gestures or gestures composed of different hand trajectories or relative positions between hands, as in the case of the system proposed for Cabral et al. for use in Virtual CAVE Environments [Cabral et al. 2005] (actually, Cabral's system uses 2D trajectories, while Wii can track in 3D, but the limitation to trajectories or relative positions is the same). Differences in hand pose cannot be accounted in such systems (recognition of gestures combining trajectories and pose usually require either complex computer vision algorithms or the use of datagloves considered awkward by many users), limiting the space of possible gestures. With the WiiMote, associating trajectory-based gestures and button inputs may reduce this limitation, but not overcome it.

Regardless of this limitation, the WiiMote, as a 3D device either for tangible or gesture-based interfaces, can be useful to enJine, to explore 3D interfaces during certain courses, to keep enJine up-to-date with game technologies, for student motivation or simply to explore the limits of these new interfaces within games or Augmented Reality.

The next section discusses related work, before section 3 discusses enJine (specially its input layer) and 4, the WiiMote. Section 5 explains in detail the solution adopted to integrate them (illustrated in Figure 1). Finally, section 6 presents the results of tests and prototypes based on this solution and the final section shows conclusions derived from this work.

2. Related Work

Nintendo's Wii and its controller have been available to the public for a relatively short time, therefore there is not much published work to be found about its use. Before presenting this work, however, this section discusses some underlying questions, such as tangible and gesture-based interfaces (particularly the problem of gesture recognition with the WiiMote) and games that use them. Architectures with flexible input layers,

like enJine, that could probably make use of the WiiMote just as simply, are also described.

Tangible Interfaces were defined previously as those where there is a correspondence between a real and a virtual object. This allows the user to manipulate the virtual objects in 3D through touch and manipulation of the real object, in a very natural manner. Compare, for instance, the task of rotating a real object with one's own hands to get the same movement in a three-dimensional digital model and of to using a mouse to select the virtual object and then using some complex interface to rotate it in three axes. Tangible interfaces also allow and even stimulate collaboration between users by allowing movement and interaction with fewer constraints, with both hands and even the body. The "ownership" of a virtual object, for instance, can be instantly determined by seeing which user holds its physical correspondent.

In TUI applications, it is important to keep registration between physical and virtual objects. There are many ways to accomplish this. The WiiMote uses acceleration sensors. Another common solution is the use of computer vision and fiducial markers (distinctive visual markers on objects) as is done by the popular ARToolKit [Billinghurst et al. 2000].

For tangible interfaces to become intuitive and seamless, with little need of training for their manipulation, it is necessary to choose physical objects that are already well-known to the targeted users (that is probably one of the reasons why the Wii controller so closely resembles a TV remote) and also make use of metaphors that appear natural to these users, thus eliminating the need to learn new ways of virtual interaction, as users only have to rely on abilities naturally developed during the course of their lives.

Gesture-based interfaces have been researched for even longer than TUI, but only in the past couple of decades enough computing power has been more accessible to bring them to reality, and it is an area where intense research is done to this day. It is curious that after all this time, it is entertainment that is popularizing this sort of interface.

Gesture-based systems require two main tasks to be accomplished: analysis and recognition. Analysis is the task of gathering data about the gesture and is usually accomplished either through active position, bending or acceleration sensors in the user's hands (datagloves are a common example) or through computer vision techniques, an usually more complex process, but which frees the user from any awkward devices. Computer vision-based analysis is especially adequate for AR applications, since a video camera is already often used in these applications when combining real and virtual images. Gesture recognition is a particular problem in the field of pattern recognition. Given the data acquired during the analysis phase, the system usually must classify the gesture as one element of a known set. The problem is complicated due to the variations that exist in every gesture, even when performed by the same user. Statistical and Artificial Intelligence (AI) techniques are often used for recognition.

Reviewing the extensive body of work in this area is beyond the scope of this paper, especially when there are good reviews available about gesture-based interfaces in general [Huang and Pavlovic 1995; Marcel, 2002] or for vision-based systems [Pavlovic et al. 1997; Wu and Huang 1999]. Of particular interest, however, is how these tasks are performed by Nintendo's Wii. A company specializing in AI for entertainment, AiLive, was approached by Nintendo and developed a tool called LiveMove [AiLive 2007] that uses context learning, a machine learning technique, to interpret Wii motion data as gestures. Game developers build a "motion recognizer" for each gesture they wish to classify in a given game by providing training samples of the gesture and can, therefore, use gesture recognition in a flexible and efficient way without needing expertise in the complex field of motion interpretation. AiLive claims the main advantage of this tool is its performance. It is able to recognize up to 40 different gestures on 8 Wii remotes or nunchuks simultaneously at 60 frames per second using less than 5% of the Wii's modest CPU power and typically less than 700kb of memory. This proprietary solution can be either supplied by Nintendo to authorized Wii developers as part of the development toolkit or acquired directly from AiLive and supposedly works with other motion-sensing controllers, as well as with MS Windows XP.

For open-source and didactic work, as in the case of enJine, using black-box proprietary tools such as LiveMove is much less advantageous than for the game industry, however, and developing alternatives to LiveMove is not only a necessity, but also an interesting future work. Works that discuss how to obtain accurate position measures from WiiMote's acceleration data, for instance using Kalman Filters [Rasco 2007], are useful in that direction.

Long before Nintendo's Wii and WiiMote, researches saw the potential of 3D, natural interfaces in games, particularly in Augmented Reality games and related several projects that use tangible interfaces or gestures. Here, only a few examples of these projects are presented, to illustrate their different approaches. The CAT/STARS platform [Magerkurth et al. 2004], False Prophets [Mandryk et al. 2002] and Monkey Bridge [Barakonyi et al. 2005] use tangible interfaces in board-like electronic games, respectively using RFID tags, infrared diodes and fiducial markers. AR sports games, such as the Golf Simulator [Govil et al. 2000] often use sport equipment itself, such as a golf club, as tangible interfaces. AcquaGauntlet [Oshima et al. 1999] uses sensor-based gesture recognition as input while AR PushPush [Kim et al. 2005] uses computer vision to recognize simple, static gestures.

It is also important to review other existing solutions to adapting novel input devices to existing software systems. Two different approaches to this problem can be identified: the creation of an abstraction layer to decouple input devices from stimuli processing and the emulation of well-known input devices through an adaptation layer.

The CIDA architecture [Farias et al. 2006] is an example of the input abstraction layer approach. CIDA defines a set of input device classes based in physical characteristics of those devices. An application developer using CIDA interfaces with one of those abstract classes, instead of directly handling input from the physical devices. This way, any input device with a suitable plug-in for that device class can be used in the application. The ICON system [Dragicevic and Fekete 2004] also employs a device class model for input devices, although it is more focused on the representation of user interactions through multiple devices. The Haptik library [Pascale and Prattichizzo 2005] defines an abstraction layer based for a specific class of input devices. Much like CIDA, devices in Haptik are represented by software components that communicate with the host application through a standard interface that exposes the device attributes. The advantage of this approach is that the application becomes inherently ready for new input devices. It also encourages programming the user interface based on intended user actions, instead of specific physical inputs. However, using an abstract input layer requires learning a new programming library and its conceptual model, which may increase development time and program complexity.

The GlovePIE software [GlovePIE 2007] follows the emulation approach. Essentially, that software is capable of reading input data from several different input devices and generating events to the operating system as if they had come from a keyboard, mouse or joystick. A gesture captured through a virtual reality glove, for instance, could be used to generate a mouse button press event. The conversion between incoming input signals and outgoing events is controlled by a custom script language. An advantage to this approach is that it is possible to use new input devices in software that was written to support only conventional devices such as keyboard and mouse. Furthermore, there is no need to learn a new programming library to handle different input devices. However, if a scripting language is used, it must be learned, and may be limited in the input signal conversions that can be expressed. Another disadvantage to this approach is the introduction of additional processing, due to the signal conversion and event generation.

Finally, there are a couple published works that relate the use of WiiMote. Curiously, both use it in musical training applications. A Master's Dissertation analyzes its use for interactive percussion instruction [Belcher 2007] and describes interesting details of the WiiMote's workings and WiiRemoteFramework, part of the WiiLi project [WiiLi 2007]. The Pinocchio system [Bruegge et al. 2007], on the other hand, allows the use of the WiiMote as an input device to conduct a virtual orchestra.

3. EnJine

EnJine is an open-source game engine based on the Java 3D API, intended to serve as a didactic tool and described in detail in other works [Nakamura et al. 2006]. EnJine is aimed at teaching game design and computer science, especially computer graphics [Tori et al. 2006] and software engineering subjects, to undergraduate or graduate students. It is also used as a test bed for research in game development, virtual and augmented reality and other related fields. The software and additional documentation are available at http://enjine.incubadora.fapesp.br/portal. Some of its chief features are:

- 3D rendering based on Java 3D (which, in turn, currently uses OpenGL);
- Multi-stage collision detection (currently only implementing subspace and bounding volume filters) and ray-casting collision detection;
- An abstract input layer to simplify the use of non-conventional input devices;
- Core classes constituting a game's basic architecture: the game, its stages, game objects (which are very flexible) etc.;
- A framework package to facilitate the creation of certain games (currently only single-player games);
- Separation between graphical and logical update cycles.

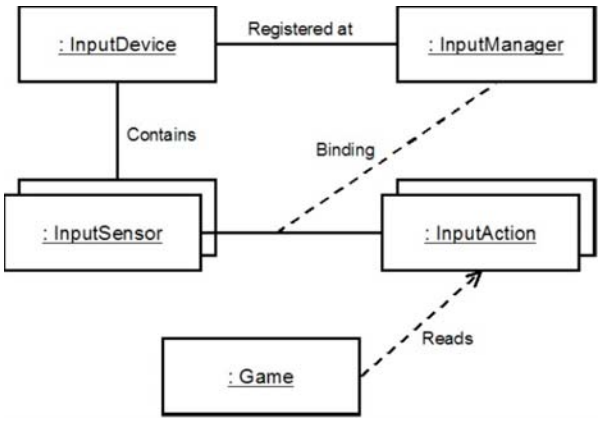


Figure 2. The input abstraction architecture in EnJine

In this paper, particular attention needs to be given to enJine's input layer. It implements a simple architecture for input abstraction, presented in Figure 2. Different input devices are accessed as concrete implementations of the `InputDevice` class, which in turn exposes its capabilities as a collection of `InputSensor` objects. Each `InputSensor` has a numeric identifier, a description and a floating-point value that can be used for both discrete-state sensors, such as buttons, as well as multi-valued sensors such as

position trackers. Each `InputDevice` implementation is responsible for updating the state of its sensors.

The `InputManager` class maintains a registry of `InputDevices` and can be queried by the user application. It also maintains a collection of "bindings" that relate each `InputSensor` to an `InputAction`. `InputActions` are objects that represent user commands with a specific intent within the application. For instance, there might be "jump" and "run" `InputActions`. Periodically, the `InputManager` updates the state of the `InputActions` according to the state of the `InputSensors`. The state of the `InputActions` may be accessed by other game components to execute desired user commands.

This simple yet flexible input layer was created at first merely to simplify the configuration of game input options in conventional devices and to accommodate these devices but has proven particularly adequate to interfacing with non-conventional devices, such as cameras tracking fiducial markers, color-marked objects or gestures. Currently, 3 projects extend enJine in this direction: integrating enJine and jARToolKit, Robot Arena and integrating enJine and HandVu.

The first project using non-conventional interfaces and enJine was to integrate it with jARToolKit (a port of ARToolKit to Java) and creating several game prototypes, including a somewhat complex dance game named "We AR Dancin'", to test the possibilities of using fiducial markers and computer vision as input devices for AR games [Tsuda et al. 2007]. These tests revealed several possibilities, limitations and particularities of this sort of interaction, such as the importance of visual feedback to the user. While during these tests the fiducial markers were mostly used for registration purposes in simple gesture-based interfaces, the resulting system can just as easily be used with markers on tangible interfaces. Aside from enabling enJine to aid when teaching 3D interfaces and AR concepts (such as its definition and the concept of registration), at a lower level of abstraction this work allows the exploration of image processing concepts (for instance, to remove video background).

Robot ARena [Calife et al. 2007] is an infrastructure that can be used as a platform for development of innovative indoor Augmented Reality games based on tangible interfaces or a wireless controlled robot and Spatial Augmented Reality (SAR), which brings the visualization of the augmented content directly into the physical space [Bimber and Raskar 2005]. The main goal of this project was to develop and test that infrastructure, exploring new ways of combining several techniques of low-cost tracking (such as by color, infrared or ultrasound), robot control and SAR. All these subjects can be explored didactically by this extension of enJine.

Finally, a work currently in progress aims to integrate enJine and HandVu [Kolsch and Turk 2005], a real time gesture recognition library able to handle both hand tracking (for trajectory-based gestures) and to recognize 6 different hand poses. The main didactic purpose of this work is to study the potential and limitations of gesture-based interfaces, especially when it is possible to recognize not only movement but also different hand poses.

4. WiiMote Specifications

Nintendo aimed to match the familiarity of a remote control and motion-sensing technology to make gaming accessible to a broader audience and the result was the WiiMote, considered the most important reason for Wii's success. The remote also includes a speaker, a rumble feature and 4 blue LED's for feedback as well as an expansion port for additional input devices, such as the Nunchuk, which are used in conjunction with the controller. The Nunchuk contains the same motion-sensing technology enabled in the Wii Remote but also includes an analog stick to assist in character movement. Using Bluetooth technology, WiiMote sends user actions to the console from as far as 10 meters away. As a pointing device, it can send a signal from as far as 5 meters away. Up to 4 Wii Remotes (plus 4 accessories such as the Nunchuk) can be connected at once and the controller and accessories can be used ambidextrously [Nintendo 2007].

According to the WiiLi Project [WiiLi 2007], an unofficial but rather complete source of information about the WiiMote, it follows the Bluetooth Human Interface Device (HID) standard and sends reports to the host with a maximum frequency of 100Hz. The controller does not require any of the authentication or encryption features of the Bluetooth standard. In order to interface with it, one must first put the controller into *discoverable* mode by either pressing specific buttons. There are 12 buttons on the WiiMote. Four of them are arranged into a directional pad, and the rest are spread over the controller. When a button is pressed or released, a packet is sent to the host via a specific HID input report containing the current state of all the buttons. This state is also included in all other input reports. The motion of the remote is sensed by a 3-axis linear accelerometer capable of measuring accelerations over a range of ±3g with 10% sensitivity. The sensor uses a right-handed coordinate system with the positive X-axis to the left, and the positive Z-axis pointing upward, when the remote is held horizontally, as show in Figure 1. Inputs on each axis are digitized to 8 bit unsigned integers. WiiMote's motion sensors are not accurate enough to use the remote to control an on-screen cursor. To correct this, Nintendo augmented the controller with an infrared image sensor on the front, designed to locate two IR beacons within the controller's field of view. The beacons are housed within the "Sensor Bar". WiiLi also documents the complete list of report types and the corresponding data transmitted in each report. Currently, the sound

data format that must be sent to the speakers is WiiMote's least known aspect to the general public.

5. Implementation

To integrate enJine and the WiiMote, the first task was to find a way to connect and communicate with the wireless controller. The first attempt was through the use of Bluetooth. This approach, however, met with several difficulties. First, while J2ME faces a less problematic situation due to the JSR 82 Bluetooth specification [Thompson et al. 2006], for PC applications using the JSE there is no implemented official Bluetooth API for Java yet. Several proprietary (and not free) APIs exist [Java Bluetooth.com 2007], but using such solutions is unthinkable for extending enJine with its open-source and free nature. An excellent library for Java, Avetana [Avetana 2007], is only free for a 14 day trial or in Linux. Trying some of the free Bluetooth communication libraries, both for Java and for C/C++, led to a few disappointments. Bluesock [Bluesock 2007], for instance, works fine with RFCOMM devices, but not with USB Bluetooth dongles using other protocols. Even Windows-specific solutions were considered, using the Bluetooth management and sockets support for Windows. However, while studying this solution, a simpler (but still platform specific) alternative was found.

The solution was to leave Bluetooth communication and management for the operating system (in this case, Windows) to manage, since it can manage all Bluetooth protocols, profiles and stacks transparently. In Figure 1 this is represented by the lowest, gray layer. This solution first requires, therefore, Windows to recognize the WiiMote as a device and connect to it (as mentioned before, for this particular work a USB Bluetooth dongle was connected to handle communication). This is not as straightforward as it may seem, and a detailed tutorial was written after some experimentation and with the aid of WiiLi. This tutorial is available in enJine's homepage.

Once the WiiMote is connected via the operating system, it can be treated as any other device, regardless of the nature of its connection. It is possible, then, to write a simple driver for it, using the Windows Device Driver Kit (or DDK). Figure 1 shows this driver as the blue layer. It is written in C/C++ and implements 3 basic functions: `initialize` finds the available Bluetooth devices and reserves some memory to be used during the communication while `read` and `write` are responsible for the actual transfer of data between the WiiMote and the computer (more details about the usage of these functions are documented along the source code, which can be found in enJine's homepage). Using JNI, this functionality is encapsulated in a DLL and made available to Java applications. Currently, to simplify implementation, this driver has some limitations: first, it is specifically built to communicate only with the WiiMote - the size of the reports used by `read` and `write` are hardcoded to the size used by Wii (22 bits). Modifying the code to allow a varying report size would be relatively simple and would allow the driver to be used with any Bluetooth device. A second limitation is that the initialize function only searches for one device (it stops searching after the first is found) and therefore multiple devices cannot be handled. This is also a relatively simple modification that will be addressed in future works. The C/C++ communication driver code was based on the cWiimote library [Forbes 2007].

This driver provides, by design, only low level, general functionality, however. A Java class named `WiiMote` (represented by the yellow layer in Figure 1) is then added to the architecture to parse, interpret and store this data. To read data from the controller, the user must first call the method `setReportType` to set the kind of information the controller must send. After that is done, each call to the `read` method requests this data from the WiiMote, interprets it and stores it as attributes that represent the last controller's state and can be read through access functions. To modify WiiMote's outputs (such as rumble or LED states) a similar procedure is followed: `setReportType` is called and attributes representing the states to be changed are modified through access functions, then calling `updateOutput` sends this data to the WiiMote.

This is all that is necessary to connect the WiiMote with enJine (or any Java application). From here on it is a matter of extending the functionality of enJine's input layer by creating `InputDevices` for the WiiMote. These devices, illustrated in pink in Figure 1, are specific to the each game and may range from the very simple, where each `InputSensor` reflects raw data representing the controller's state, to sophisticate devices that use pattern recognition to interpret user movements and associates each `InputSensor` to a given gesture. The game (purple in Figure 1) then associates these sensors to `InputActions`, just as it would do with a conventional device like a keyboard, and treats these actions normally.

To test the solution's implementation as well as to serve as proofs of concept and ready-made devices that can be used directly or as models by students or developers, two simple devices were developed. One implements a very simple gesture-based interface while the other functions as a TUI. Both devices work simply with acceleration, without integrating it or using complex pattern recognition techniques. This simplicity is actually important, to avoid confusing a new user who tires to understand the basics of how such a device works before trying anything more sophisticate. Both implemented devices also contain an update thread that periodically queries the WiiMote about its state. These threads conflict with each other when accessing the same controller, so it is currently impossible to combine two `InputDevices` to obtain a

more complex functionality. Each of these devices can be understood as a different style of interaction.

The first device is called `WiiImpulseDevice` and translates strong, fast movements with the controller as directional commands. Flicking the WiiMote up activates the "up" `InputSensor` and the same goes for right, left and down. When returning the control to a rest, default position (i.e. horizontal, face-up and pointing forward) the movement is less brisk and does not "undo" the last motion. A problem arises in this device due to working purely with acceleration data, instead of velocity or position: every time the controller is flicked in a direction, it suffers a strong acceleration in that direction, but also an even stronger deceleration to make it stop. If acceleration data was used directly, this would immediately "undo" the last input intended by the user. To address this matter, `WiiImpulseDevice` ignores a negative acceleration peak right after a positive peak (i.e. a positive movement) and vice-versa.

The second device, called `WiiTiltDevice`, simply assumes that only slow, gentle movements are applied to the WiiMote, so that gravity is by far the strongest acceleration (actually, the acceleration caused by force applied by the user's hand that resists gravity - it has the opposite sign) to calculate the controller's pitch and roll based on how gravity is decomposed in the X and Y axes. Yaw is not measured (it would require the use of the infrared sensor). `WiiTiltDevice` could be used for games requiring subtle and continuous motions for control, such as controlling the flight path of a glider through its inclination. To avoid calling the `asin` (which would be necessary to determine tilt) method at such a high update rate as WiiMote's, sin is linearized as $\sin \alpha = \alpha$. This is reasonable for low values of α but becomes a bad approximation as it increases. At higher angle values, however, the WiiMote accelerometers themselves loose resolution as their direction grows closer to gravity's and high angles should, therefore, be avoided for this particular device anyway. The approach of using asin in the calculations was actually implemented, tested and even found acceptable, although the lag it causes even in a simple application was noticeable (but not severe).

Two simple prototype applications were built with enJine to test these two devices. They merely draw a ColorCube on a black background (that is why such a cube is shown as the prototype in Figure 1) and have its movement controlled by `WiiImpulseDevice` or `WiiTiltDevice`. Finally, a previously existing, simple enJine game was modified to use as its input `WiiImpulseDevice` or `WiiTiltDevice` (a rather simple modification, simply replacing in the code the names of the old keyboard devices and sensors with the new ones - the actions and the rest of the game remain the same). In this action puzzle, called RollCubes and show in Figure 3, several ColorCubes exist in a plane and up to two users may control one at a time with a cursor (the cones in Figure 3), rolling it up, down, left or right on its faces. If two or more cubes become adjacent with the same face up (each face has a different color), all of them begin to slowly disappear. The goal is to clear all ColorCubes as quickly as possible.

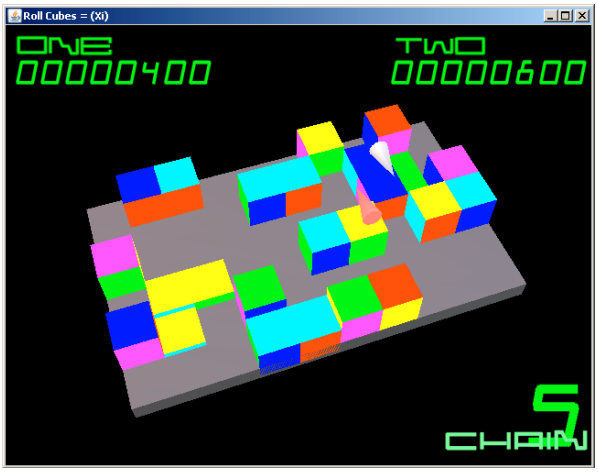


Figure 3: RollCubes

6. Results

The first piece of data tested in this work was WiiMote's refresh rate. It is listed nominally as 100Hz and it was necessary not only to be sure of this number, but also to verify if the integration with enJine and all those layers did not lower this rate significantly. It was indeed confirmed that the refresh rate is 100Hz (this was the value used in all further tests and devices) and that the integration with enJine had not lowered it. When attempting to query data at a faster rate, especially immediately after initialization, caused errors (but no crashes or exceptions). It was decided, then, that for all InputDevices using the WiiMote, the thread polling the controller's state would sleep for 10ms (at least) between each query.

Before implementing the devices and prototypes described in section 5, a simple application that records raw data from the WiiMote was also built, to analyze this data. Some of the most useful analyses came from the controller in a static position, because then only a known acceleration (1g) is applied to it. Figure 4 shows its response to gravity in 10ms intervals over a period of 5s. In the first 2s, the controller is resting on it's back on a flat surface, thus the positive acceleration in the Z axis, indicating the vertical force that is being applied on the WiiMote by the surface to resist gravity. Then the controller is turned to rest on its left side and the force is applied along the X axis. It's interesting to notice that while this apparent inversion of the sign for measured accelerations happens only with resisted field-generated forces, such as gravity. For forces directly applied, by the user's hand for instance, the acceleration is measured with the expected sign.

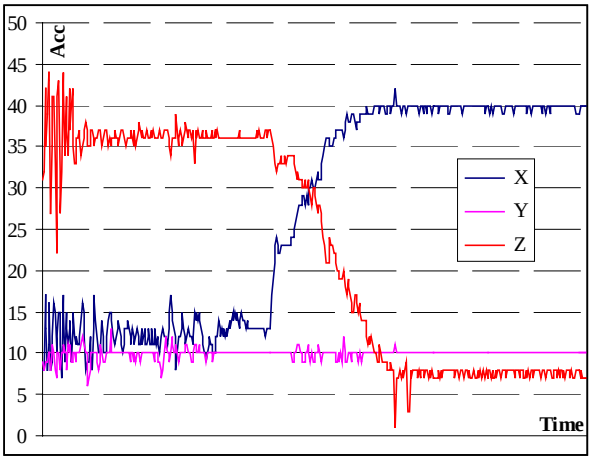


Figure 4. WiiMote's response to gravity

From this test, the offsets in measured acceleration for each axis and the value of g can also be determined. For this particular controller, they are approximately 12, 10 and 8 for the X, Y and Z axes, respectively and g corresponds to 28. In the 8-bit word used to record accelerations by WiiMote, the range of possible values goes from -127 to 128, which is more than enough to accommodate the nominal 3g it can measure. These values for offsets and gravity vary for each controller and can be obtained through a specific HID report.

Another fact that can be noticed in Figure 4 is how errors in the measures of acceleration are relatively small when the controller is in repose, but can be considerably high (even higher than the 10% nominal sensitivity) during or right after movements. Finally, this test can also show the directions of the positive axes, which were found to correspond to those described by WiiLi and in section 4.

Figure 5 shows the accelerations measured by the WiiMote during 3.5s, while a single, wide arm swing was applied from right to left, with the controller held horizontally and pointing forward. The acceleration in the Z axis (pointing up) corresponds to the resistance to Gravity. In the X axis there is first a positive acceleration pushing the controller from left to right, then an even higher deceleration to make it stop (this behavior was discussed previously and is taken in consideration in `WiiImpulseDevice`). This was a "weak" swing, thus the low positive acceleration in X. The most intense (and interesting) acceleration, however, appears along the Y axis, which points to the user. So the arm swing can make the controller move in an arc (as in fact happened), this radial acceleration must be applied. It corresponds to the centripetal force responsible for the circular motion. It is interesting to notice that as soon as the circular movement stops, this acceleration reaches zero. Unlike for linear movements, there is no need for a deceleration to match the centripetal force before the controller can reach repose. The small variation in the Z acceleration has the same behavior as the X acceleration (i.e. it corresponds to a linear movement) and indicates that the movement was not perfectly parallel to the X axis: either the controller was slightly tilted around the Y axis during the movement (a strong possibility) or the arm movement had a vertical component, as well as an horizontal one.

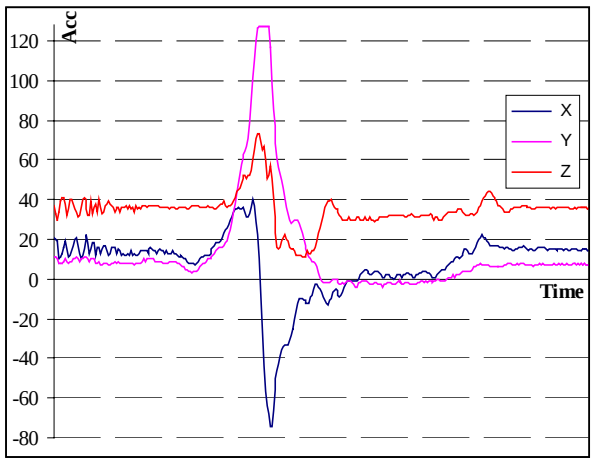


Figure 5. Accelerations during a swinging movement

These tests led to the `InputDevices` described in section 5. The implementation of `WiiImpulseDevice` was quite straightforward after understanding Figure 5. On the other hand, while inspired by Figure 4, implementing `WiiTiltDevice` was more complicated due to the need to decompose the acceleration along the axes of the moving coordinate system. Both devices worked as designed during testing.

No usability tests were made for the game modified to use `WiiImpulseDevice` or `WiiTiltDevice`. Informally, however, it can be noted that using `WiiImpulseDevice` in a game that requires several quick and repetitive movements is hard on the user's wrist but using `WiiTiltDevice` led to a pleasant interface, but still slower than simply using the keyboard. Then again, modifying this game to use the WiiMote was not accomplished to improve the game's playability (it is actually the sort of game that works with conventional devices very well), but simply to analyze how much coding work is required to replace a conventional device with a pre-made WiiMote `InputDevice` and the result was positive: the work involved is very small, as discussed in section 5.

7. Conclusions

This paper described the integration of the WiiMote and enJine, successfully achieving the proposed goal. This integration (as well as other projects such as Robot ARena and the integration with ARToolKit and HandVu described previously) serves the double purpose of allowing the exploration of innovative game technologies for didactic and research purposes, as well as keeping enJine up-to-date with the trend of 3D, more natural interfaces in games. This not only increases student motivation, but also expands the

possibilities to use enJine in teaching, particularly for courses involving AR, user interfaces or games.

The integration solution chosen has multiple layers and, at the lowest level, makes use of the operating system's handling of Bluetooth communication, since finding a comprehensive and free library for this function was problematic. A more efficient solution was to write a simple device driver to the WiiMote using the Windows DDK and allowing Java to access it using JNI. This driver comprises the second layer of the architecture. The third layer, already in Java, is a class that interprets WiiMote's raw data, stores its state as attributes and provides a simple interface to send outputs to the controller. With this functionality, using the WiiMote becomes just as simple as using any other device. Two sample `InputDevices` where created and an existing game was modified to use them.

Testing helped ascertain the unofficial data about the WiiMote, such as its refresh rate and coordinate system, as well as showed that the implemented architecture did not impair this refresh rate significantly. It was found that, while in repose or under low accelerations, errors in acceleration measures by the controller were relatively small, but under higher accelerations they were significant (even surpassing the 10% nominal sensor sensitivity). Testing was also important to understand and visualize WiiMote's response to user movements to implement `WiiImpulseDevice` and `WiiTiltDevice`.

While the project's goal was satisfactorily reached, the implementation still has a few limitations. Perhaps the most important is the dependence of the Windows operating system. Because Bluetooth communication was isolated in the lowers layers of the proposed architecture, however, once a suitable and free Java API for this task is found (or, in a worst case scenario, developed internally), it should be relatively simple to use it to make the system platform-independent. Until then, however, the Windows system will be prioritized over other solutions because, in the particular case of this work's authors, it is used much more commonly in classes and by the students.

Other limitation that can also be addressed in future works is to decouple the device driver from the WiiMote, allowing it to serve as a low-level communication interface for any Bluetooth device. Perhaps even more important is to modify the system to work with more than one controller at a time. Adding the possibility of combining more than one `InputDevice`, to allow the creation of complex interfaces built with simpler modules, is also an interesting future work. Another possibility to study is extending this work to include other motion sensing controllers, such as Sony's Sixaxis wireless controller.

Perhaps the most interesting possible future works with this system are using it to explore pattern recognition to build more complex gesture-based interfaces and putting the system to practical use, both using the system in classrooms and building games with innovative 3D interfaces (particularly gesture-based and TUIs). To this end, a game is currently being developed combining enJine, the WiiMote and Robot ARena. While enJine is more of an educational tool in itself, instead of a tool for making educational games, using the WiiMote makes the concepts of accelerations and forces very tangible and building an application to teach those concepts is another interesting future work.

Finally, this work was more concerned with the necessary infrastructure for integrating WiiMote and enJine than with studying the controller's use, but formally studying usability aspects of games using WiiMote is an interesting and necessary future work.

References

TORI, R.; BERNARDES, J. AND NAKAMURA, R., 2006. Teaching Introductory Computer Graphics Using Java 3D, Games and Customized Software: a Brazilian Experience. *In: Digital proceedings of the 34th International Conference and Exhibition on Computer Graphics and Interactive Techniques, Educators Program*. ACM SIGGRAPH.

GOMES, P. AND PAMPLONA, V., 2005. M3GE: um motor de jogos 3D para dispositivos móveis com suporte a Mobile 3D Graphics API. In: Proceedings of the Brazilian Symposium on Computer Games and Electronic Entertainment 2005. SBC, 55-65.

NINTENDO Wii Remote. http://wii.nintendo.com/controller.jsp [accessed 20 August 2007].

DOBSON, J., 2006. Nintendo: 600,000 Wii, 454,000 Zelda Sold In First Eight Days. Available on: http://www.gamasutra.com/php-bin/news_index.php?story=11864 [accessed 20 August 2007].

CARLESS, S., 2005. Xbox CFO Estimates Initial Xbox 360 Sales. Available on: http://www.gamasutra.com/php-bin/news_index.php?story=7109%3C/link%3E [accessed 20 August 2007].

AZUMA, R., 1997. A Survey of Augmented Reality. Presence: Teleoperators and Virtual Environments, n. 6, 355-385.

ISHII, H. AND ULLMER, B., 1997. Tangible Bits: Towards Seamless Interfaces Between People, Bits And Atoms. *In: Proceedings of The SIGCHI Conference on Human Factors In Computing Systems*, 234-241.

CABRAL, M.; MORIMOTO, C. AND ZUFFO, M., 2005. On the usability of gesture interfaces in virtual reality environments. *In: Proceedings of the 2005 Latin American conference on Human-computer interaction*. ACM, 100-108.

BILLINGHURST, M.; KATO, H. AND POUPYREV, I., 2000. Artoolkit: A computer vision based augmented reality toolkit. *In: Proceedings of the IEEE Virtual Reality 2000 Conference*. IEEE.

HUANG, T. AND PAVLOVIC, V., 1995. Hand Gesture Modelling, Analysis and Synthesis. In: Proceedings of the International Workshop on Automatic Face And Gesture Recognition 1995. IEEE, 73-79.

MARCEL, S. Gestures for Multi-Modal Interfaces: A Review. 2002. Available on: http://citeseer.ist.psu.edu/marcel02gestures.html [accessed 20 August 2007].

PAVLOVIC, V.; SHARMA, R. AND HUANG, T., 1997. Visual Interpretation of Hand Gestures for Human Computer Interaction: A Review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 19, n. 7, p. 677-695.

WU, W.; HUANG, T., 1999. Vision-Based Gesture Recognition: A Review. Springer. 103p.

AILIVE, 2007. LiveMove White Paper.. Available at: http://www.ikuni.com/papers/LiveMoveWhitePaper_en.pdf .

RASCO, B., 2007. Where's the Wiimote? Using Kalman Filtering To Extract Accelerometer Data. Available on: http://www.gamasutra.com/view/feature/1494/wheres_the_wiimote_using_kalman_.php [accessed 20 August 2007].

MAGERKURTH, C.; ENGELKE, T. & MEMISOGLU, M., 2004. Augmenting the Virtual Domain with Physical and Social Elements. *In: ACM SIGCHI Advances in Computer Entertainment*. ACM, 163-172.

MANDRYK, R.; MARANAN, D. & INKPEN, K., 2002. False Prophets: Exploring Hybrid Board/Video Games. *In: Proceedings of the Conference of Human Factors in Computing Systems 2002*. 640-641.

BARAKONYI, I.; MARKUS W.; THOMAS P. AND SCHMALSTIEG D., 2005. MonkeyBridge: Autonomous Agents in Augmented Reality Games. *In: Proceedings of the ACM SIGCHI International Conference on Advances in Computer Entertainment Technology 2005*. ACM.

GOVIL, A.; YOU, S. AND NEUMANN, U., 2000. A Video-Based Augmented Reality Golf Simulator. In: Proceedings of the 8th ACM International Conference On Multimedia. ACM.

OHSHIMA, T.; SATOH, K.; YAMAMOTO, H. & TAMURA, H., 1999 RV-Border Guards: A Multi-player Entertainment in Mixed Reality Space. *In: Proceedings of the IEEE and ACM Second International Workshop on Augmented Reality*. ACM.

KIM, K. ET AL., 2005. ARPushPush: Augmented Reality Game in Indoor Environment. *In: Proceedings of the International Workshop on Pervasive Gaming Applications 2005*.

FARIAS, T.; TEIXEIRA, J.; RODRIGUES, C.; PESSOA, S.; COSTA, N.; TEICHRIEB, V. AND KELNER, J., 2006. CIDA: an interaction devices management platform. *In: Proceedings of the Symposium on Virtual Reality*. SBC, 271-284.

DRAGICEVIC, P. AND FEKETE, J., 2004. Support for input adaptability in the ICON toolkit. *In: Proceedings of the 6th international conference on Multimodal interfaces 2004*. ACM, 212-219.

PASCALE, M. AND PRATTICHIZZO, D., 2005. The haptik library: a component based architecture for uniform access to haptic devices. *IEEE Robotics and Automation Magazine*. (in press.)

GLOVEPIE. http://carl.kenner.googlepages.com/glovepie [accessed 20 August 2007].

BELCHER, J., 2007. *Embodied Interfaces for Interactive Percussion Instruction*. Master's Dissertation. Available on: http://scholar.lib.vt.edu/theses/ available/etd-5312007-113346/ [accessed 20 August 2007].

WIILI. http://www.wiili.org [accessed 20 August 2007].

BRUEGGE, B. ET AL., 2007 Pinocchio: conducting a virtual symphony orchestra. *In: Proceedings of the international conference on Advances in computer entertainment technology 2007*. ACM, 294-295.

NAKAMURA, R.; BERNARDES, J.L.; TORI, R., 2006. enJine: Architecture and Application of an Open-Source Didactic Game Engine. *In: Digital Proceedings of the Brazilian Symposium on Games and Digital Entertainment 2006* Available on http://www.cin.ufpe.br/~sbgames/proceedings/index.htm [accessed 20 August 2007].

TSUDA, F.; HOKAMA, P.; RODRIGUES, T. AND BERNARDES, J., 2007. Integration of jARToolKit and enJine: extending with AR the potential use of a didactic game engine. *In: Proceedings of the IX Simposium on Virtual and Augmented Reality*. SBC, 51-59.

CALIFE, D.; TOMOYOSE, A.; SPINOLA, D. BERNARDES JR.; J.L. AND TORI, R., 2007. Robot ARena: Infrastructure for Applications Involving Spatial Augmented Reality and Robots. *In: Proceedings of the IX Symposium on Virtual and Augmented Reality*.

BIMBER, O. AND RASKAR, R., 2005. Spatial Augmented Reality: Merging Real and Virtual Worlds. Wellesley: A K Peters.

KOLSCH M. & TURK, M., 2005. Hand Tracking with Flocks of Features. *In: Proceedings of 2005 Conference on Computer Vision and Pattern Recognition.* IEEE.

THOMPSON ET AL., 2006. JSR 82. Available on: http://jcp.org/en/jsr/detail?id=82 [accessed 20 August 2007].

JAVA BLUETOOTH.COM, 2007. Development Kits. Available on: http://www.javabluetooth.com/development_kits.html [accessed 20 August 2007].

AVETANA, 2007. Available on: http://www.avetana-gmbh.de/avetana-gmbh/produkte/jsr82.eng.xml [accessed 20 August 2007].

BLUESOCK, 2007. Available on: http://bluesock.dev.java.net/ [accessed 20 August 2007].

FORBES, K., 2007. cWiimote. Available on: http://simulatedcomicproduct.com [accessed 20 August 2007].

Um *Framework Opensource* para a Construção de Sistemas Multiatores

Allan D. S. Lima Patricia R. Tedesco Geber L. Ramalho

Universidade Federal de Pernambuco, Centro de Informática, Brasil

Palavras-chave: Atores Sintéticos, Sistemas Multiagentes, Simuladores, Sistemas Multiatores.

Contado dos Autores:
{adsl, pcart, glr}@cin.ufpe.br

Resumo

Com a popularização dos sistemas multiagentes surgiram diversos *frameworks* para dar suporte ao seu desenvolvimento, reduzindo o custo e tempo de desenvolvimento desses sistemas. Além disto, também há vários projetos, dentre até jogos, que utilizam múltiplos atores sintéticos, um tipo específico de agente inteligente que possui personalidade, atua em ambientes multimídias ou virtuais e são representados graficamente por avatares. Entretanto, os projetos que fazem uso de múltiplos atores sintéticos não costumam utilizar os *frameworks* existentes durante a sua implementação. Isto se deve ao fato de que esses *frameworks* não costumam se aplicar tão bem a sistemas multiatores pois não dão suporte as características diferenciais destes sistemas. Assim, este trabalho apresenta o *FAct* (*Framework for Actors*), um projeto criado especificamente para auxiliar na construção sistemas multiatores, visando reduzir o custo e tempo de desenvolvimento para os projetos baseados nesta tecnologia.

1. Introdução

Aplicados à resolução de problemas em diversas áreas, como Educação a Distância [Torreão et al. 2005], Comércio Eletrônico [Guttman et al. 1998] e Jogos [Barreteau et al. 2001], agentes inteligentes são entidades computacionais com duas importantes características: autonomia e sociabilidade. Estas propriedades permitem a construção de Sistemas Multiagentes (SMAs) que apresentam comportamento flexível e dinâmico. No entanto, como SMAs são sistemas complexos, comumente seu desenvolvimento é baseado em *frameworks*. Estes agrupam as funcionalidades básicas necessárias a SMAs, como controle do ciclo de vida dos agentes e transporte de mensagens, permitindo ao desenvolvedor se preocupar apenas com a modelagem comportamental dos agentes, o que reduz significativamente o custo de desenvolvimento dos projetos.

No entanto, com a utilização cada vez maior de agentes inteligentes em sistemas que necessitam engajar o usuário na sua interação, vem-se destacando a modelagem de um tipo especial de agente, denominado ator sintético (também chamados de personagens virtuais). Estes são agentes inteligentes credíveis, que apresentam personalidades em um estilo particular de interação, percepção, raciocínio e ação [Silva et al. 2000].

Este tipo diferenciado de agente vem sendo utilizado no desenvolvimento de diversos projetos de pesquisa (e.g. [Bates et al. 1994; Rousseau e Hayes-Roth 1997])e também aplicados na construção de jogos [Silva et al. 2000; SmartSim 2007] visando enriquecer a credibilidade dos agentes e melhorar a qualidade da interação com o usuário. No entanto, estes personagens tendem a ser cada vez mais complexos, o que torna o custo e tempo de desenvolvimento destes projetos bastante expressivos.

Em contrapartida, os *frameworks* para a construção de SMAs atuam justamente no suporte aos SMAs reduzindo o custo e o tempo de desenvolvimento deles. Entretanto, projetos que aplicam atores sintéticos não fazem uso de qualquer tipo de *framework para SMAs*. Isto se deve ao fato que de que tais bibliotecas não dão suporte às características particulares dos atores, como por exemplo, o suporte a modelos de personalidade. Isto inviabiliza a aplicação dos mesmos para a construção de projetos baseados em atores.

Neste contexto, este artigo apresenta o *FAct* (*Framework for Actors*), que oferece um *framework open source* para construção SMAct. Ele possui uma biblioteca para ajudar no desenvolvimento da simualção e um conjunto ferramentas gráficas de suporte ao desenvolviment, que podem ser utilizados e estendidos pelo desenvolvedor, que se encarregará apenas da implementação do comportamento dos atores e das demais entidades presentes na simulação, aumentando assim a sua produtividade e, conseqüentemente, reduzindo o custo e o tempo de desenvolvimento dos SMAct.

O presente artigo está organizado em cinco seções. A segunda apresenta um estudo comparativo entre projetos que utilizam o conceito de atores sintéticos; a terceira faz uma análise crítica dos *frameworks* para a construção de SMAs; a quarta apresenta o conceito de Sistemas Multiatores (SMAct), discutindo algumas diferenças importantes entre SMAs e SMAct; a quinta descreve o *FAct* abordando seus requisitos, seus módulos, sua arquitetura e a aplicação exemplo desenvolvida sobre ele; a sexta, por fim, relata as

conclusões do projeto e mostra os próximos passos no desenvolvimento do projeto.

2. Atores Sintéticos

Atores sintéticos, ou agentes credíveis, são entidades computacionais que possuem como objetivo principal transmitir "ilusão de vida" aos usuários de suas aplicações. Para que isto seja possível, eles contam com atributos que possuem uma semântica diferenciada que podem representar os estado emocional do agente, a sua personalidade além de avatar para o mesmo[Silva et al. 2000], além das demais características que um agente autônomo possui, como, por exemplo, os serviços que ele provê, seu estado interno e um mecanismo de comunicação.

No decorrer dos últimos anos, a utilização de atores sintéticos vem se popularizando. Sendo estes aplicados para simulação do comportamento de indivíduos em diversos projetos. Além disto, sua utilização nestes projetos quase sempre envolve a aplicação de mais do que um ator em uma única simulação, compondo sistemas com múltiplos atores. Como exemplos, podemos citar os projetos Oz [Bates et al. 1994], Teatro Virtual [Rosseau e Hayes-Roth. 1997], além do projetos Enigmas no Campus [Silva et al. 2000], VICTEC [Paiva et al. 2004] e *SmartSim* [SmartSim 2007]. Os dois primeiros são analisados pelo pioneirismo na aplicação de modelos de personalidade baseados em psicologia aliados a agentes autônomos. Já os dois últimos são analisados, pois tiveram como resultado a o desenvolvimento de jogos baseados em atores sintéticos.

A seguir, cada um destes projetos é abordado com mais detalhes sob o ponto de vista dos seus *objetivos*, *suas aplicações* e *seus modelos de personalidade e emoção*.

O projeto OZ

O projeto Oz [Bates et al. 1994] foi um dos primeiros na utilização de atores sintéticos, tendo sido idealizado para simulação de ambientes artisticamente interessantes. Uma das aplicações desenvolvidas no projeto o OZ foi o *Lyotard* que é uma ambiente de texto para simulação do comportamento de um gato em seu ambiente cotidiano. Outra aplicação criada a partir deste projeto simula o comportamento de três crianças em um playground. Em ambas, utilizou-se um modelo de personalidade que tem como base (1) a curiosidade, (2) o contentamento, (3) a capacidade de ignorar, (4) a agressividade, (5) a amigabilidade e (6) a arrogância/orgulho. Este modelo é utilizado para determinar o comportamento dos atores com o auxílio de um modelo de emoções baseado no modelo OCC [Orthony et al. 1990].

Teatro Virtual

O projeto Teatro Virtural [Rousseau 1997] foi criado para desenvolver atores sintéticos que podem improvisar seus comportamentos em ambientes interativos sem planejamento detalhado. O objetivo do projeto é a construção de atores que interagem entre si e com os humanos para criar novas histórias cada vez que são utilizados. Dentre as aplicações desenvolvidas durante o projeto, podemos ressaltar o *Cybercafe,* que simula a interação entre os clientes, controlados por humanos, e um garçom que é um ator sintético, e o simulador multiator *Animated Puppets*, um mundo gráfico habitado por criaturas que dialogam e brincam umas com as outras. Estas criaturas têm seus comportamentos baseados na Teoria dos Traços [Allport 1966] e na Teoria do Aprendizado Social [Bandura 1977]. A primeira assume que o perfil de uma personalidade individual pode ser descrito em termos de um conjunto de traços psicológicos constantes que influenciam o comportamento. Entretanto, na segunda teoria, a personalidade é modificada a cada situação que é vista como uma experiência de aprendizado por reforço.

Enigmas no Campus

O Projeto Enigmas no Campus [Silva et al. 2000] é um jogo multiagentes de aventura em que o jogador deve solucionar enigmas a fim de restituir o conhecimento do Universo destruído por criaturas maldosas e alienígenas. Ele foi criado para testar, não só um modelo psicossocial desenvolvido por pesquisadores da UFPE, mas também para avaliar a aplicação uma arquitetura para atores sintéticos e outros agentes em jogos de aventura criada igualmente na UFPE. Seu modelo de personalidade é baseado na teoria *The Big Five* [Howard e Howard 1995] que classifica a personalidade em cinco tipos distintos: extrovertido, aprazível, consciente, neurótico e "aberto à experiência".

SmartSim

O projeto *SmartSim* [SmartSim 2007], criado na UFPE, tem como objetivo principal o desenvolvimento de um *framework* de código aberto para a construção de jogos de negócios [Faria e Nulsen 1996] que utilizem atores sintéticos. Como primeira aplicação do *framework* foi desenvolvido o protótipo de um jogo sério chamado *Virtual Team*, que tem como objetivo apoiar a capacitação de gerentes de projeto de software com base nos conceitos do PMBOK® [PMBOK 2004]. O comportamento dos personagens neste projeto é baseado no *Symlog* [Bales e Cohen 1979] que classifica a personalidade do indivíduo em três dimensões Dominância(U)/Submissividade(D), Amigável(P)/Não Amigável(N) e Aceitação(A)/Não aceitação de autoridade e orientação à tarefa (B) [Silva et al. 2006].

Discussão

Todos os quatro projetos estudados possuem aplicações nas quais mais de um ator sintético é utilizado. Além disto, estes projetos têm um foco no desenvolvimento de arquiteturas para a implementação de atores, ou a experimentação de modelos de personalidade simulados no computador. Neste contexto, apenas o projeto *SmartSim* se diferencia, pois ele é focado no desenvolvimento de um *framework* para a construção de jogos de negócios.

Contudo, nenhum dos projetos propõe algo em relação à arquitetura das aplicações desenvolvidas. De forma geral, eles se preocupam apenas em testar modelos personalidade aplicados na simulação de um determinado contexto. Porém, diversos aspectos invariantes nestes projetos, como o suporte à comunicação entre os atores e o controle do ciclo de vida dos mesmos, foram implementados de maneira similar em cada um, sem reutilizar nada dos projetos anteriores. Por exemplo, os atores sintéticos do jogo *Virtual Team*, foram desenvolvidos sem o auxílio de qualquer biblioteca ou ferramenta que já suportasse as características comuns aos sistemas com múltiplos atores, pois estas simplesmente não existiam.

Apesar de não haver projetos que auxiliem especificamente no desenvolvimento de sistemas multiatores, é possível encontrar diversos *frameworks* para o desenvolvimento de sistemas multiagentes.

3. Projetos para a construção de simuladores multiagentes

Durante o processo de desenvolvimento do *FAct* diversos projetos para construção de SMAs foram avaliados sob cinco critérios: (1) Objetivos; (2) Implementação e disponibilidade de código fonte; (3) Documentação do projeto; (4) Ferramentas de suporte ao desenvolvimento; e (5) Quantidade e domínio de suas aplicações. De acordo com os critérios listados, três projetos se destacaram: o CORMAS [Page et al. 2000], o JADE [Bittencourt e Osório 2002], o SeSAm [Klügl e Puppe 1998].

CORMAS

Criado para simular interações entre um grupo de agentes e um ambiente que guarda recursos naturais, o *Common–pool Resources and Multiagent Systems* (CORMAS) [Page et al. 2000] foi implementado em *Smalltalk* [Abdala e Wangenheim 2002], e seu código fonte é distribuído sob uma licença pública própria.

A diversidade de aplicações desenvolvidas com o projeto é um dos seus diferenciais e uma conseqüência da sua utilização em diversas instituições de pesquisa ao redor do mundo, para simular desde a dinâmica de uma savana até o equilíbrio em florestas do Mediterrâneo. Dentre suas ferramentas se destacam um construtor de modelos de simulação além de um compacto gerenciador de simulação.

JADE

O *Java Agent DEvelopment Framework* (JADE) [Bellifemine et al. 1999] foi criado com o intuito de ser um ambiente para desenvolvimento de aplicações baseadas em agentes inteligentes conforme as especificações da *Foundation for Intelligent Physical Agents* (FIPA) [FIPA 2007].

Implementado em Java [Deitel e Deitel 2005], ele é distribuído sob a *Lesser General Public License* [LGPL 2007]. E, como o CORMAS, é utilizado em diversas aplicações (e.g. na construção de jogos multiagentes [Morais e Furtado 2003]). Além disso, sua principal ferramenta é um depurador para o comportamento dos agentes.

SeSAm

o SeSAm [Klügl e Puppe 1998] provê um ambiente genérico para modelagem e experimentação de sistemas baseados em agentes. Implementado em Java, este projeto tem o seu código fonte distribuído sob a licença LGPL, assim como o JADE.

Um dos principais diferenciais do SeSAm é a quantidade de ferramentas que ele possui. Foram implementadas diversas aplicações para dar suporte ao *framework*, a saber: (1) uma ferramenta para construção de modelos de agentes; (2) uma interface para modelagem de animações; e (3) um programa para o desenvolvimento do protocolo de comunicação dos agentes, possibilitando ao programador implementar boa parte da simulação sem escrever uma linha de código

Discussão

Distribuídos livremente, o JADE, o SeSAm e o CORMAS têm uma grande comunidade de desenvolvedores que trabalham não só contribuindo diretamente com o projeto, mas também no desenvolvimento de aplicações sobre eles.

Não por acaso, o JADE, o CORMAS e o SeSAm, também possuem uma vasta documentação em suas páginas. Dentre estes, podemos destacar o SeSAm que possui uma Wiki onde os membros do projeto podem escrever de maneira colaborativa informações fundamentais sobre como instalar, configurar, compilar e estender o ambiente.

Os três projetos também se destacam pelo número de ferramentas para dar suporte ao seu desenvolvimento. Neste ponto, o JADE merece ênfase, pois o seu acervo de programas de apoio ao desenvolvimento é bastante vasto, permitindo até a execução remota e passo a passo da simulação, o que

facilita consideravelmente o processo de testes das aplicações.

Quanto à tecnologia de implementação destes projetos, o JADE e o SeSAm usam Java, enquanto que o CORMAS foi implementado em *Smaltalk*. Nos três casos a tecnologia de implementação adotada acaba por dificultar a aplicação dos mesmos na construção de jogos, já que C++ [Stroustrup 1997] é a linguagem de programação mais utilizada nestes tipos de aplicações [Rolling e Morris 2004].

Outra característica interessante é que o JADE possui uma preocupação, em sua arquitetura, com os padrões da FIPA, o que pode facilitar consideravelmente a interoperabilidade entre sistemas multiagentes distintos.

4. Sistemas Multiatores (SMact)

A análise realizada na seção dois mostrou que todos os projetos estudados possuem aplicações onde mais de um ator sintético está presente. O Projeto Oz simula diversas crianças em um playground, o Projeto Teatro Virtual utiliza diversos atores no *Cibercafe*, o Enigmas no Campus também faz uso de múltiplos atores em seu ambiente, e o *SmartSim* os utiliza na simulação do comportamento de desenvolvedores de *software*.

Isso se justifica, pois o emprego de múltiplos atores pode ser muito útil, incrementando a interatividade da aplicação através da simulação do comportamento dos atores baseados em diversos tipos de personalidade e emoções.

Por exemplo, no jogo *Virtual Team* o usuário tem que escolher os desenvolvedores que irão compor uma equipe de desenvolvimento de software. O desempenho da equipe selecionada depende, entre outros fatores, das personalidades de seus membros. Outra conseqüência da utilização de múltiplos atores é o estabelecimento de um nível de relacionamento entre os agentes. Isto também pode ser verificado no *Virtual Team*, como uma das variáveis relevantes para o desempenho da equipe na simulação. Através da interface do jogo é possível acompanhar todo o processo de simulação, inclusive estado emocional dos atores nos seus avatares.

4.1 Uma definição para Sistemas Multiatores

Tendo como fundamento a deliberação para sistemas multiagentes proposta por Ferber (1999) e as características de um ator sintético, é possível definir o termo 'sistema multiatores' como aplicável a um sistema com os seguintes elementos:

1. Um ambiente, E, representando um espaço que geralmente tem um volume;
2. Um conjunto de objetos, O. Estes objetos são geograficamente situados, e em qualquer dado momento é possível associar qualquer objeto com uma posição em E. Estes objetos podem ser percebidos, criados, destruídos e modificados pelos atores;
3. Um conjunto de atores sintéticos, A, que são objetos específicos (A está contido em O), representando as entidades ativas do sistema;
4. Um conjunto de representações gráficas, RG, para os atores A, tal que cada ator possua pelo menos uma representação;
5. Um conjunto de relações R, que liga objetos (e seus agentes) entre si;
6. Um conjunto de operações, Op, tornando possível para os agentes de A perceber, produzir, consumir, transformar e manipular objetos de O;
7. Operadores com a tarefa de representar as reações do ambiente a Op.

Os pontos 3 e 4 desta definição representam as principais modificações que foram realizadas na definição original para SMAs. O ponto 3 indica a existência de um conjunto de atores sintéticos, o que requer a concepção de uma coleção de agentes com todas as características que ator sintético possui, emoções, personalidade, etc. Já o ponto 4 ressalta a necessidade de um conjunto de representações gráficas para os atores. Elas fazem parte da interface gráfica do projeto, e representam o estado atual do agente para o usuário da aplicação.

5. O *FAct: Framework for Actors*

Como vimos anteriormente, a análise dos projetos para a construção de sistemas multiagentes mostrou que a maioria deles não restringe os tipos de agentes aos quais pretendem apoiar, visando desta forma, auxiliar no desenvolvimento de praticamente qualquer tipo de agente inteligente, sem, entretanto, suportar características de tipos específicos de agentes, como os atores sintéticos. Em decorrência, desta generalidade, projetos como o Enigma no Campus [Silva et at. 2000] e o *SmartSim* [SmartSim 2007] optaram por realizar toda implementação dos seus ambientes de simulação sem o auxílio de qualquer tipo de *framework* multiagente. Isto se deve ao fato de que os frameworks atuais para multiagentes não dão suporte as características peculiares aos atores sintéticos, como por exemplo, o suporte a modelos de personalidade.

Visando preencher esta lacuna, o *FAct* (Framework for Actors), um *framework* para construção de Sistemas Multiatores foi elaborado. O *FAct* tem como principal objetisavo apoiar o desenvolvimento de

SMAct em projetos que, assim como o Enigma no Campus e o *SmartSim*, teriam que implementar os seus agentes sem o suporte de ferramenta alguma.

5.1 Requisitos Elicitados

Como forma de definir o escopo de um *framework* para a construção de sistemas multiatores, um processo de elicitação de requisitos foi iniciado. Este processo teve como base a análise dos projetos que utilizam atores sintéticos nos seus ambientes de simulação, as ferramentas para a construção de sistemas multiagentes e a definição criada para sistemas multiatores. Ao final deste processo, seis requisitos fundamentais foram identificados. A seguir, cada um destes requisitos é descrito em detalhes.

[RQ01] Fornecer um modelo genérico e extensível tanto para atores sintéticos quanto para suas representações gráficas

Normalmente, os projetos para simuladores multiagentes provêem um modelo genérico para construção de agentes. Este modelo também deve estar presente em um *framework* para implementação de sistemas multiatores. Entretanto, existem características particulares dos atores como a sua personalidade, que também devem ser levadas em consideração em sua construção. Visando atender outra característica particular de um ator sintético, a sua representação gráfica, um *framework* para construção de SMAct também deve fornecer um modelo genérico para tais representações. Além disto, estes modelos, de ator e de representação, devem possibilitar que o desenvolvedor possa estendê-los de acordo com as necessidades da aplicação que está desenvolvendo.

[RQ02] Controlar o ciclo de vida dos atores e o de suas representações

De maneira similar aos *frameworks* para implementação de SMAs, também se faz necessária a criação, o gerenciamento e a posterior destruição dos atores em um *framework* para construção de SMAct, porém este suporte também deve ser estendido para as representações gráficas dos atores. Desta forma, o desenvolvedor precisa apenas se preocupar em definir o seu modelo de ator além do seu comportamento enquanto que o *framework* se encarrega de controlar o ciclo de vida destes.

[RQ03] Prover mecanismos genéricos e extensíveis para a comunicação entre os atores

Como forma de possibilitar a interação entre os atores durante a simulação, se faz necessária a construção de um modelo genérico e extensível de comunicação. Este modelo deve possuir definições para as mensagens trocadas, mecanismos que possibilitem a troca destas mensagens além de meios para divulgação e busca dos serviços que cada ator provê [Burg et al. 2004].

[RQ04] Fornecer mecanismos para análise da simulação

Importantes para o controle, a avaliação e a compreensão do processo de simulação, estes mecanismos permitem não só a depuração desta por parte do desenvolvedor, mas também um maior entendimento da simulação por parte do usuário que a executa. Neste contexto, é fundamental a implementação de uma ferramenta capaz de interpretar os arquivos de *log* gerando gráficos e de tabelas a partir destes.

[RQ05] Ser distribuído sob uma licença *open source*

Um dos fatores que influenciam na criação de uma comunidade de desenvolvedores que contribuem em projeto é a sua licença de distribuição. Sendo assim, este requisito indica que um *framework* para construção de SMAct deve preferencialmente ser *open source* além de ser distribuído sobre uma licença difundida na comunidade.

[RQ06] Ser implementado em C++

Segundo Bittencourt (2002), Java é a tecnologia favorita para a construção de SMAs, pois, 66.04% deles são implementados nesta linguagem. Entretanto, isto acaba por reduzir a aplicabilidade destes projetos em áreas como o desenvolvimento de jogos, onde C++ é predominante [Rolling e Morris 2004]. Por isto, a implementação do *framework* para construção de SMAct em C++ pode vir a aumentar o campo de aplicação destes sistemas. Esta restrição se aplica a todo o *framework*, exceto às ferramentas de suporte a simulação, pois estas devem ser independentes da biblioteca de componentes que é utilizada na implementação do código fonte da simulação.

5.2 Módulos do *FAct*

Visando contemplar os requisitos do *framework*, um conjunto de componentes, foi eliciado para compor o FAct. Basicamente o projeto se divide entre o *Framework de Simulação* e as *Ferramentas de suporte*, como mostrado na Figura 1. O primeiro é composto por um conjunto de componentes que podem ser reutilizados na implementação do código-fonte da aplicação. O segundo é um conjunto de aplicações para auxílio no desenvolvimento e depuração da simulação.

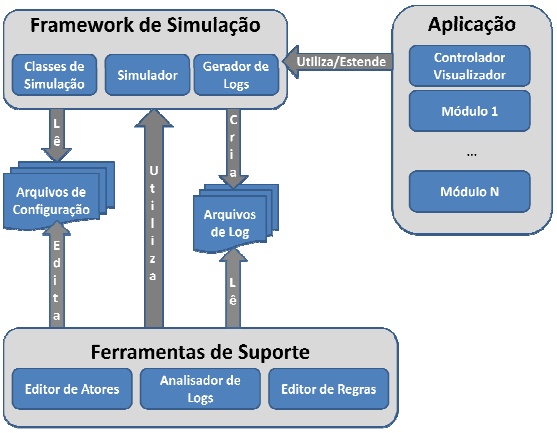


Figura 1 - Visão geral do *FAct*

Framework de Simulação

Responsável pela base da simulação este módulo é subdivido em três componentes, a saber: (1) Classes de Simulação, (2) Simulador e (3) Gerador de Logs. A seguir cada um destes é apresentado detalhadamente.

Classes de Simulação: Compõem uma biblioteca de classes com o objetivo de dar suporte à implementação da simulação. As classes de simulação compreendem o modelo de ator sintético além do modelo de mensagem que é utilizado na comunicação dos agentes. Estas classes são as entidades básicas necessárias para a construção de simuladores e devem permitir facilmente modificações e extensões. Este módulo atende aos requisitos RQ01 e RQ03.

Simulador: Implementando os requisitos RQ02 e RQ03, este módulo é composto por um conjunto de classes responsável por gerenciar e executar a simulação além de conter os mecanismos de comunicação.

Gerador de Logs: Visando arquivar um histórico das mensagens e eventos gerados pela simulação, este módulo pode ser utilizado tanto para a análise posterior da simulação, como por outras ferramentas. Este módulo atende o requisito RQ04.

Controlador/Visualizador: Tem como principal objetivo permitir ao usuário executar, pausar, ajustar a velocidade e visualizar os eventos da simulação. A implementação deste módulo depende do domínio de cada aplicação desenvolvido sobre o *FAct* e está relacionada ao requisito RQ05.

Ferramentas de Suporte

Composto por um conjunto de aplicações para auxiliar no processo de desenvolvimento, depuração e análise da simulação, este módulo possui três ferramentas, a saber: (1) editor para os atributos do atores; (2) editor para as regras de simulação; (3) uma ferramenta de análise dos logs gerados durante a simulação.

Editor de Atores: Ferramenta responsável por permitir ao usuário do simulador a configuração dos atributos dos atores, tais como personalidade e emoções, que estarão presentes no ambiente de simulação. Este editor atua como uma interface gráfica para facilitar o processo de modelagem dos atores contribuindo para a implementação do requisito RQ01.

Editor de Regras: Interface gráfica composta por um editor de regras e fórmulas para a simulação do ambiente. Esta ferramenta facilita a modelagem do mundo onde a simulação ocorrerá e contribui para a implementação do RQ01.

Analisador de Logs: Fazendo parte da implementação do RQ05, este módulo é composto por uma aplicação capaz de interpretar os arquivos de log da simulação para a geração de tabelas e gráficos permitindo o acompanhamento do comportamento dos atores durante a simulação.

5.3 Implementação

Composta por 5 pacotes (Figura 2), a arquitetura do *FAct* foi projetada e implementada para atender aos requisitos identificados na seção 4. Atualmente a arquitetura já contempla os módulos Classes de Simulação, Simulador, e Gerador de *Log*. Fica então, restando o Controlador/Visualizador que deve ser desenvolvido de acordo com cada aplicação a ser criada sobre o *framework*. Além disto, o módulo de edição de atores foi desenvolvido como uma aplicação independente das demais partes do projeto e já se encontra plenamente funcional. Por fim, analisador de *logs* tem a sua finalização prevista para o fim do ano corrente.

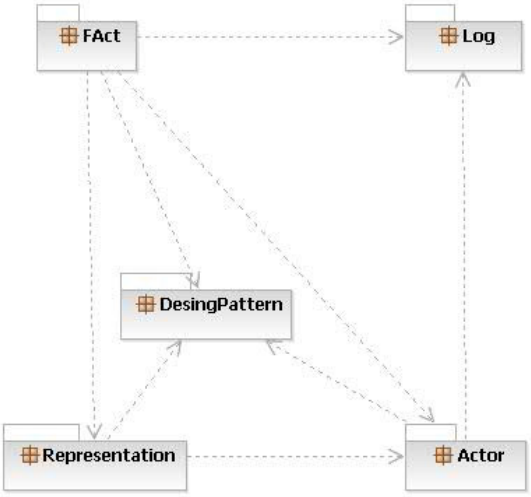


Figura 2: Pacotes do *framework* e suas dependências.

O pacote *Actor*

Contendo o modelo genérico de ator sintético bem como o modelo genérico de mensagem para a comunicação dos atores, o pacote *Actor* atende aos RQ01 e RQ04 do projeto compondo assim, a implementação das classes de simulação.

O modelo ator desenvolvido dá suporte à criação de atributos para a representação das características presentes no ator. Eles podem conter, por exemplo, o modelo de personalidade adotado para a simulação. A implementação deste modelo genérico se dá através de uma árvore que guarda as informações referentes ao atributo, como o seu nome, seu valor, e a sua descrição.

Além disso, este pacote possui um gerenciador de comunicação que permite o envio de mensagens entre os atores, compondo assim as Classes de Simulação e implementando o RQ04 do *framework*.

Este pacote contém ainda classes que possuem as funções de controlar o ciclo de vida dos atores, gerenciar os serviços que cada ator possui e gerenciar o processo de simulação. O que também o torna parte da implementação do segundo e do terceiro requisitos, além de compor o módulo Simulador.

O pacote *Representation*

Como atores sintéticos devem ser representados graficamente por avatares, este pacote foi criado especificamente para isto. Ele faz parte das Classes de Simulação e da implementação do RQ03. Além de possuir uma classe genérica e extensível para representar o avatar de um ator, ele também é responsável pelo controle do ciclo de vida destes avatares. A dependência do pacote *Actor* se deve ao fato que as representações estão diretamente associadas ao atores.

O pacote *Log*

O gerador de *logs* do *FAct* foi implementado neste pacote. Ele possui classes responsáveis por controlar e representar as mensagens de *log* geradas durante todo o processo de simulação.

É possível criar diversos arquivos de log; além disso, a habilitação/desabilitação do sistema de *log* pode ser realizada a qualquer momento.

O pacote *FAct*

Servindo como uma fronteira entre a aplicação e o *framework*, este pacote contém uma classe que implementa o padrão de projeto *Facade* [Gamma et al. 1998]. Os métodos dessa classe retornam objetos encapsulados por interfaces que possuem a implementação dos serviços que o *FAct* provê. Daí o fato destes pacote depender de todos os demais. Por fim, o isolamento dos módulos que se dá através de interfaces é um ponto fundamental para permitir a manutenção do *framework*, além da sua própria forma de desenvolvimento incremental.

O pacote *DesignPattern*

Auxiliando aos demais, este pacote possui a implementação genérica de alguns padrões de projeto, a saber: (1) *Iterator*, *Singlenton* e *AbstractFactory* [Gamma et al. 1998].

O padrão *Iterator* é utilizado pelas classes que possuem coleções de objetos dentro do *FAct*. A utilização deste padrão permite que as coleções sejam percorridas de forma seqüencial, um objeto a cada interação. Já o padrão *Singleton* é utilizado na implementação das classes do projeto que devem possuir apenas um objeto instanciado durante a execução da simulação. As classes que possuem tal propriedade devem herdar da implementação de genérica, presente no *framework,* evitando a redundância de código. Finalmente, o padrão *AbstractFactory* é utilizado pela classe fachada do *FAct* para construir os objetos que representam o serviços do *FAct,* tal classe é feita.

Da mesma forma que este pacote foi utilizado pelo próprio *framework*, ele também é passível de utilização pelos desenvolvedores quando eles estiverem construindo suas aplicações.

A Ferramenta de Edição de Atores(?)

Criada para atender a demanda identificada de modelar um ator através de uma interface gráfica, esta ferramenta (Figura 3) permite ao desenvolvedor descrever cada atributo das instâncias dos atores que compõem a sua simulação.

A criação de atores pode ser realizada de três maneiras distintas. O novo ator pode ser baseado em um ator vazio (sem atributos), em outro ator previamente modelado ou em um modelo de personalidade. Atualmente a ferramenta dá suporte aos modelos MBTI [Myers et al. 1998], *Symlog* [Bales e Cohen 1979], *The Big Five* [Howard e Howard 1995] e Eysenck [Eysenck 2005].

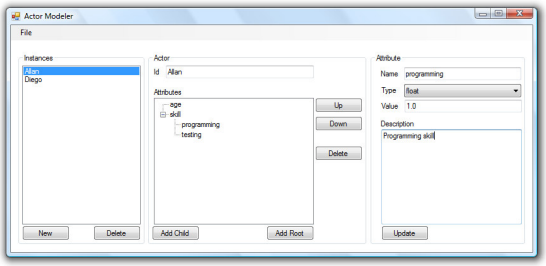


Figura 3: Interface da ferramenta de modelagem de atores.

Através desta ferramenta, atributos são definidos em uma árvore onde cada nó representa um atributo

como mostrado na Figura 4. Cada atributo possui um nome, um tipo, um valor e uma descrição. Os modelos criados podem ser salvos em arquivos no formato XML [Hunter et al. 2007] e posteriormente carregados em tempo de execução pelo *FAct* ou por qualquer outra aplicação seja capaz de ler o modelo.

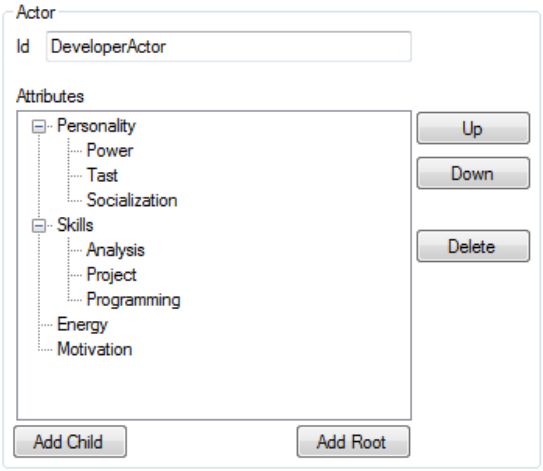


Figura 4: Exemplo de um arvore com os atributos de um ator.

5.4 *MD-Simulation*: uma aplicação desenvolvida sobre o *FAct*

Como forma de testar a aplicabilidade do *FAct* na construção sistemas multiatores foi desenvolvido *M-D Simulation*, um simulador multiatores criado a partir do *FAct*. O M-D Simulation trata de interações entre um gerente de projeto e um desenvolvedor de software, conforme mostrado na Figura 5.

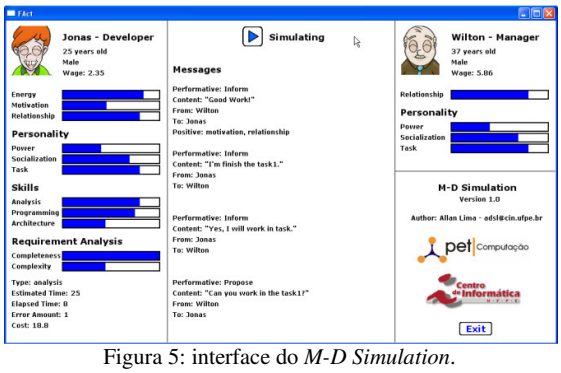


Figura 5: interface do *M-D Simulation*.

Assim como nos projeto *SmartSim*, o modelo de personalidade implementado nos atores foi baseado na teoria *Symlog* [Bales e Cohen 1979]. Com base neste modelo dois atores foram implementados, o *Developer Actor* e o *Manager Actor*. O primeiro simula o comportamento de um desenvolvedor de software. O Quadro 1 mostra os atributos que ele possui. Compondo a simulação do comportamento deste ator, quatro ações foram modeladas para ele. São elas: (1) pedir descanso, (2) pedir ajuda para realizar uma tarefa, (3) trabalhar em uma tarefa e (4) reportar a conclusão de uma tarefa.

Quadro 1: Propriedades do *Developer Actor*.

Atributo	**Descrição**
Skills	**Habilidades técnicas do ator**
Analysis	Capacidade de Elicitação/Análise de Requisitos
Architecture	Aptidão para o projeto de Arquitetura de um software
Programming	Competência para implementação (codificação)
Personality	**Modelo de personalidade baseado no Symlog**
Power	Dominância(U)/Submissividade(D)
Socialization	Amigável(P)/Não Amigável(N)
Task	Aceitação(A)/Não aceitação de autoridade(B)
Others	
Energy	Energia do ator, mede o quanto ele esta cansado
Motivation	Força que impulsiona as ações ator
Name	Nome do ator
Age	Idade do ator

Modelado como versão simplificada do *Developer Actor*, o *Manager Actor* é responsável pela simulação do comportamento de um gerente de um projeto. Ele possui a maioria dos atributos presentes no *Developer Actor* exceto pelas habilidades técnicas (*Skills*), motivação (*Motivation*) e energia (*Energy*). Entretanto, seu comportamento é completamente distinto. Suas ações são (1) delegar uma tarefa, (2) mandar o desenvolvedor descansar, (3) ajudar o desenvolvedor em uma tarefa, (3) dar opinião sobre uma tarefa realizada, (4) demitir o desenvolvedor.

Com relação ao emprego do *FAct* no desenvolvimento desta aplicação, suas classes foram úteis em diversos pontos. Primeiramente, as classes que representam os atores da simulação herdam do modelo genérico presente no *framework*. Desta forma, as propriedades que ambos possuem são armazenadas na árvore de atributos citada anteriormente.

Outra funcionalidade utilizada no *FAct*é o suporte a páginas brancas e amarelas. Assim, quando um ator é criado no *MD-Simluation*, ele guarda os serviços que oferece, uma funcionalidade suportada pelo *framework*, além de registrá-los no gerenciador de serviços que também já está presente no *FAct*. Isto facilita consideravelmente, por exemplo, a implementação da interação "pedir ajuda", pois o ator que necessita de ajuda solicita ao gerenciador de serviços um ator capaz de fornecer tal serviço e, em seguida, só precisa iniciar o processo comunicação com o mesmo.

A comunicação entre os atores também é realizada através do FAct. Inicialmente, todas as mensagens trocas entre os atores são instâncias da classe criada no *framework* para representar mensagens. Em seguida, elas também são enviadas através do gerenciador de comunicação desenvolvido no FAct.

Na interface gráfica da simulação, as representações gráficas dos atores estendem o modelo proposto pelo FAct e são guardadas no gerenciador de representações do framework, o que facilita consideravelmente o processo de re-pintura das representações a cada ciclo da simulação.

O gerenciador de logs foi utilizado para imprimir no console todas as mensagens trocadas durante simulação. O *design pattern Singleton* foi utilizado no desenvolvimento da classe responsável por criar os objetos da simulação.

Finalmente, a construção desta aplicação permitiu a criação de um ambiente de simulação onde foi possível verificar a aplicabilidade do *Symlog* na simulação de atores em um ambiente de desenvolvimento de software. Além disto, também foi possível avaliar a aplicabilidade do *FAct* para a construção de sistemas multiatores. Isto foi feito verificando de fato quais dos seus módulos são utilizados mais freqüentemente e se o funcionamento dos mesmos está de acordo com o esperado.

6. Conclusões e Trabalhos Futuros

A análise dos projetos que utilizam o conceito de ator sintético mostrou que eles têm como foco o exame de complexos modelos de personalidade aplicados em contextos de simulação, o que os torna bastante custosos. Além disto, também foi possível verificar que diversos aspectos que pouco se alteram nestes projetos, como por exemplo, o controle da comunicação entre os atores, foram reimplementados em cada um deles.

Em contra partida, esta preocupação na redução dos custos e do tempo de desenvolvimento está presente nos vários *frameworks* para a construção de Sistemas Multiagentes. Contudo, estas ferramentas não costumam ser utilizadas no desenvolvimento de simulações com atores sintéticos, pois não possuem suporte às características particulares deles, como por exemplo, modelos de personalidade.

Com o objetivo de suprir esta deficiência foi elaborado o *FAct* (Framework for Actors) que atualmente fornece um *framework open source* para a construção de sistemas multiatores. Ele é composto por dois módulos. O primeiro, Framework Simulação, se encontra completamente implementado e é composto provê a infra-estrutura necessária ao desenvolvimento do código da simulação. O segundo, as Ferramentas de Suporte, já possui uma ferramenta de modelagem de atores desenvolvida enquanto que as demais já estão em processo de análise de requisitos.

No atual estado de desenvolvimento do FAct, já é possível construir de aplicações multiatores com ele. O primeiro ambiente desenvolvido foi o *MD-Simulation*, que simula um conjunto de interações entre um gerente e um desenvolvedor de software.

Por fim, com o intuito de realizar um esforço na consolidação do *FAct*, se faz necessária a finalização da implementação das demais ferramentas de suporte à simulação. Além disso, para estudar a aplicabilidade do Framework é preciso implementar mais aplicações, desenvolvidas inicialmente sem o *framework* e posteriormente com utilização do *FAct*, permitindo assim, uma comparação entre as diferentes versões com relação ao esforço de desenvolvimento e as métricas do código. Outra atividade importante para a consolidação do *framework* é o estudo e posterior adição ao *framework* dos modelos de emoções mais utilizados nos projetos que fazem uso de atores sintéticos.

Referências

ATBDALA, D. D. E WANGENHEIM, VON A., 2002. *Conhecendo o SmallTalk,* Sexta edição, Visual Books.

ALLPORT, G. W., 1966. *Traits Revisited*. American Psychologist 21: 1-10.

BANDURA, A., 1977. *Social Learning Theory*. Englewood Cliffs, Prentice-Hall.

BALES, R. F. E COHEN, 1979. *Symlog, A System for the Multiple Level Observation of Groups,* first edition. Free Press.

BATES, J., LOYALL E A. REILLY, W., 1994. AN ARCHITECTURE FOR ACTION, EMOTION, AND SOCIAL BEHAVIOR. SCHOOL OF COMPUTER SCIENCE, CARNEGIE MELLON UNIVERSITY. PITTSBURGH, PA, USA.

BARRETEAU O., BOUSQUET, F., ATTONATY J-M., 2001. *Role-playing games for opening the black box of multi-agent systems: method and lessons of this applications to Senegal River Valley irrgated systems.* Journal of Artificial Societes and Social Simulation vol. 4, no 2.

BELLIFEMINE, F., POGGI, A. E RIMASSA, G., 1999. CSELT internal technical report. *JADE – A FIPA-compliant agent framework*. London, April.

BITTENCOURT, J. R. E OSÓRIO F. S., 2002. *Ambiente para Simulação de Múltiplos Agentes Autônomos, Cooperativos e Competitivos*. Monografia de Conclusão de Curso. UNISINOS, São Leopoldo.

DEITEL, H. E DEITEL, P., 2005. *Java Como Programar.* Sexta edição, Prentice Hall.

EYSENCK, M., 2005. Congnitive Psychology: A Student's Handbook, 5th edition, Psychology Press.

FARIA, A. J. E NULSEN, R., 1996. *Developments In Business Simulation & Experiential Exercises,* volume 23. University of Windsor e Xavier University.

FERBER, J., 1999. *Multi-agent Systems: Introduction to Distributed Artificial Intelligence,* first edition, Addison-Wesley Professional.

FIPA, 2007. *Welcome to the Foundation for Intelligent Physical Agents* [online] http://www.fipa.org [Acessado 19 de Agosto de 2007].

GAMMA, E., HELM, R., JOHNSON, R. E VILISSIDES J., 1995. *Design patterns: elements of reusable object-oriented software,* first edition. Addison-Wesley Professional.

GUTTMAN, R. H., MOUKAS, A. G. E MAES, P., 1998. *Agents as Mediators in Electronic Commerce. International Journal of Electronic Markets,* volume 8, number 1. Massachusetts Institute of Technology, MIT Media Laboratory, Cambridge, MA, USA.

HOWARD, P. J. E HOWARD, J. M., 1995. The Big Five Quick Start: An Introduction to Five-Factor Model of Personalaty for Human Resource Professionals. Center of Applied Cognitive Studies, Chalotte, NC, USA.

HUNTER, D., RAFTER, J., FAWCETT, J., VLIST E. VAN DER, AYERS, D., DUCKETT, J., WATT, A., MCKINNON L., 2007. Beginning XML, 4th Edition, Wrox.

KLÜGL, F. E PUPPE, F., 1998. *The Multi-Agent Simulation Environment SeSAm*. Dep. for Artificial Intelligence, University of Würzburg.

LGPL, 2007. *Lesser General Public Licence* [online] Disponível em: http://www.gnu.org/licenses/lgpl.html [Acessado 19 de Agosto de 2007].

MORAIS, L. A. S. E FURTADO, J. J. V., 2003. *Estudo e Desenvolvimento de Sistemas Multiagentes usando JADE: Java Agent Development framework*. Monografia de conclusão de curso. UFCE, Fortaleza.

MYERS, I. B., MCCAULLEY, M. H., QUENK, N. L., MAMMER, A. L., 1998. MBTI Manual (A guide to the development and use of the Myers Briggs type indicator), Third edition, Consulting Psychologistis Press.

The C++ Programming Language, Third edition, Addison-Wesley Professional.

ORTHONY, A., CLORE, G., L. E COLLINS, A., 1990. *The Cognitive Structure of Emotions*, reprint edition. Cambridge University Press.

PAGE, L. C., BOUSQUET, F., BAKAM, I., BAH, A. E BARON C., 2000. *CORMAS : A multiagent simulation toolkit to model natural and social dynamics at multiplescales.*

PAIVA, A., SOBREPEREZ, P., WOODS, S., ZOLL, C., 2004. *Caring for Agents and Agents that Care: Building Empathic Relations with Synthetic Agents.*

PETER S., 2005. *Practical Common Lisp (Hardcover),* first edition, Apress.

ROUSSEAU, D. E HAYES-ROTH, B., 1997. *Improvisational Synthetic Actors with Flexible Personalities.* Knowledge Systems Laboratory, Department of Computer Science, Stanford University, Stanford, California, USA.

ROLLING A. E MORRIS A., 2004. *Game Architecture and Design: A New Edition*. New Riders Publishing.

SILVA, D. R. D. RAMALHO, G. L. E TEDESCO P. C. A. R.2000. *Atores Sintéticos em Jogos de Aventura Interativos: O Projeto Enigmas no Campus*. Dissertação – Mestrado em Ciências da Computação. CIn-UFPE.

SILVA, D. R. D. S., RAMALHO, G. L., E TEDESCO, P. 2006. *Atores Sintéticos em Jogos Sérios: Uma Abordagem Baseada em Psicologia Organizacional.* Proposta de doutoramento apresentada à Universidade Federal de Pernambuco, como parte do programa de pós-graduação em Ciências da Computação, área de concentração Computação Inteligente.

SMARTSIM, 2006. *Projeto SmartSim – Simulação em gerência de projetos com atores sintéticos.* [online] Disponível em: http://cin.ufpe.br/~smartsim [Acessado 19 de Agosto de 2007].

TORREÃO, P. G. B. C., TEDESCO, P. C. A. R., MOURA, H. P. 2005. *Project Management Knowledge Learning Environment: Ambiente Inteligente de Aprendizado para Educação em Gerenciamento de Projetos*. Mestrado em Ciências da Computação. CIn-UFPE.

STROUSTRUP, B., 1997. The C++ Programming Language, Third edition, Addison-Wesley Professional.

WOOLDRIDGE, M., 2002. *An Introduction to Multiagent Systems,* first edition. Liverpool: John Wiley & Sons, Ltd.

Um Framework para o Desenvolvimento de Agentes Cognitivos em Jogos de Primeira Pessoa

Ivan M. Monteiro
Universidade Federal do Rio Grande do Sul

Debora A. Santos
Universidade Federal da Bahia

Resumo

Em jogos eletrônicos é freqüente a necessidade da existência de agentes capazes de emular o comportamento humano. Tais agentes são denominados BOTs. Neste trabalho é apresentada uma arquitetura híbrida para agentes cognitivos no contexto de jogos, juntamente com um *framework* que dá suporte ao desenvolvimento desses agentes. Essa arquitetura visa simplificar o processo de modelagem dos BOTs. Já o *framework* fornece a estrutura para implementação e reutilização das partes desses agentes.

Keywords:: Inteligência Artificial, Agentes, Arquitetura de Agentes, Framework.

Author's Contact:

immonteiro@inf.ufrgs.br
abdalla@ufba.br

1 Introdução

Os jogos eletrônicos possuem um importante papel nos campos relacionados a inteligência artificial, podendo ser utilizados como laboratórios para testar tanto teorias sobre o raciocínio humano quanto métodos de resolução de problemas[van Waveren 2001]. O desenvolvimento de BOTs[1] caracteriza bem este papel, pois nos jogos, os BOTs devem atuar como jogadores virtuais dotados de inteligência artificial tentando emular[2] seqüências de comportamentos dos jogadores humanos.

Com o avanço tecnológico, tem-se aumentado a necessidade e realismo dos jogos atuais, o que torna o desenvolvimento dos BOTs muito mais complexos. Não basta o BOT cumprir sua função básica no jogo, que é interagir com os outros jogadores dentro da lógica do jogo, ele também precisa apresentar comportamentos naturais dos jogadores humanos, como ter conhecimento do ambiente em que está inserido, guardar sua história pessoal, exibir emoção e personalidade como se ele tivesse vida própria, sendo confundido com um jogador humano. Os jogadores humanos têm-se mostrado mais satisfeitos com o jogo se eles acreditam que o personagem com quem ele interage é um jogador humano ao invés de controlado por computador.

Em jogos, os personagens podem ser controlados tanto por humanos como por computador. O primeiro é visto como mais inteligente, flexível, imprevisível, com habilidade de adaptar suas estratégias ao estado dos jogos, possibilidade de raciocinar sobre suas ações e também capaz de utilizar diferentes estratégias para cada momento do jogo. Já os controlados por computador costumam ser lembrados como personagens previsíveis e com ações pré-definidas, o que torna entediante jogar com eles[de Byl 2004].

Para atingir o nível de satisfação desejado com os BOTs, diversas técnicas em computação são utilizadas, como máquina de estados finitos e *scripting*, mas é em inteligência artificial que se concentram as principais técnicas para se desenvolver um personagem convincente, tais como: buscas heurísticas, sistemas baseados em conhecimento, lógica fuzzy, redes neurais. Entretanto, as técnicas de forma isolada são insuficientes para resolver o problema. É a combinação adequada das técnicas que traz um resultado satisfatório. Neste contexto, a modelagem de um BOT como um agente, entidade que percebe e atua sobre o ambiente, é muito importante. Primeiro, porque o BOT está captando percepções do ambiente e executando ações como um agente. Segundo, porque a modelagem em agentes facilita a integração das diversas técnicas da inteligência artificial.

Uma metodologia para o desenvolvimento de BOTs baseado em agentes, além de útil, é bastante importante. Útil porque encurta o tempo do seu desenvolvimento atacando diretamente o problema com um modelo de sua solução. Importante porque permite o estudo e a avaliação da forma de desenvolver e das técnicas possíveis de serem utilizadas.

A forma de uma arquitetura de agentes está intimamente ligada à tarefa que o agente irá executar no ambiente[Nilsson 1998]. Por isso, a escolha correta da arquitetura para implementar um BOT facilita bastante o seu desenvolvimento. Além disso, esta escolha também propicia o estabelecimento de uma metodologia para o desenvolvimento desses agentes.

A utilização de uma estrutura definida, aliada a uma arquitetura de agente, servindo de suporte para a organização e o desenvolvimento de BOTs, simplifica bastante a criação desses jogadores virtuais. Esta estrutura desejada é fornecida neste trabalho através de um *framework* para a criação de agentes em jogos de primeira pessoa. A intenção desse *framework*, nomeado de IAF , é facilitar o desenvolvimento do software, que neste caso é o programa do agente. Esta estrutura se baseia na arquitetura híbrida para agentes cognitivos proposta em [Monteiro 2005] e traz funcionalidades e componentes de reúso para o desenvolvimento desses agentes.

Com isto, este trabalho simplifica o uso de jogos eletrônicos como laboratório para pesquisas e estudos voltados à computação e também amplia as possibilidades de desenvolvimento de jogadores virtuais. A área de Inteligência Artificial é bastante beneficiada com esta aproximação, uma vez que muitas técnicas presentes nesta área podem ser diretamente aplicadas ao contexto de jogos. Outro importante ponto é que o conhecimento utilizado neste contexto de jogos tem relacionamento estreito com as simulações da robótica, o que permite que os programas de agentes possam ser testados antes de serem embarcados.

A sessão 2 apresenta alguns trabalhos relacionados, a sessão 3 expõe a arquitetura[Monteiro 2005] que é utilizada como base do desenvolvimento do IAF . Na sessão 4 o *framework* é descrito, na quinta, são feitos alguns testes e avaliações do *framework* e do desenvolvimento de BOTs e na sessão 6 são feitas as considerações finais.

2 Trabalhos Relacionados

O projeto *GameBots*[Andrew Scholer 2000][Kaminka et al. 2002] busca levar o jogo *Unreal Tournament* ao domínio de pesquisa em inteligência artificial. Associado a ele existe o projeto *JavaBots*[Adobbati et al. 2001], que fornece uma *API* para o desenvolvimento de agentes para o jogo *Unreal Tournament*. Essa *API* tem como principal objetivo permitir que desenvolvedores consigam criar BOTs sem se preocuparem em lidar com aspectos específicos do protocolo do *GameBots*.

Apesar de também utilizar o GameBots como base, este trabalho vai além de uma abstração da comunicação com o jogo, como é visto no *JavaBot*. O *framework* aqui introduzido, fornece uma implementação estrutural da arquitetura de agente apresentada em [Monteiro 2005], aplicada ao jogo Unreal Tournament. Assim, este trabalho pretende estreitar ainda mais as distâncias entre o domínio

[1] A palavra BOT surgiu da simplificação da palavra ROBOT. Este termo ficou bastante conhecido na área de jogos referindo-se a personagens que não são controlados por jogadores humanos.

[2] Freqüentemente o termo emular é confundido com simular. Uma distinção breve seria que enquanto emular refere-se a o que é feito, simular está relacionado a como foi feito.

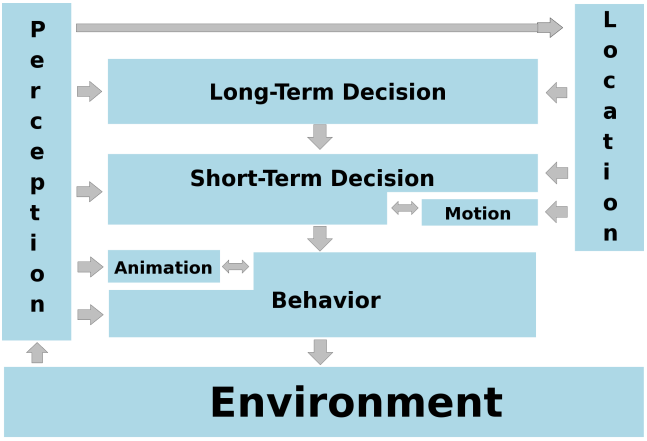


Figura 1: *A arquitetura.*

de pesquisa em inteligência artificial e sistemas multiagentes com o domínio de jogos eletrônicos.

3 A Arquitetura de Agente

Nesta sessão, é apresentada uma arquitetura modular para o desenvolvimento de agentes cognitivos em jogos de primeira e terceira pessoas[Monteiro 2005]. Essa, visa suprir uma lacuna existente na área de agentes inteligentes em jogos eletrônicos, que é a falta de arquiteturas de agente híbrido especializado ao contexto de jogos. A seguir são descritos os módulos presentes na arquitetura, juntamente com suas interações.

A arquitetura de agente cognitivo apresentada aqui é composta por sete módulos:

- **Perception -** Módulo de eventos responsável pela filtragem dos sensores do agente.
- **Location -** Centraliza informações sobre o modelo do ambiente para o agente.
- **Long-Term Decision -** Responsável pelo planejamento.
- **Short-Term Decision -** Responsável pela execução do plano atual.
- **Motion -** Manipula aspectos de roteamento, colisão e desvios de obstáculos.
- **Animation -** Módulo responsável por determinar a animação a ser exibida.
- **Behavior -** Módulo responsável pela atuação sobre o ambiente de forma reativa.

Dos sete módulos apresentados, cinco são responsáveis pelo processo de decisões do agente, são eles: *Long-Term Decision*, *Short-Term Decision*, *Motion*, *Animation*, *Behavior*. Os módulos de decisão estão agrupados em 3 camadas:

- **Camada Deliberativa -** Responsável por gerar soluções globais para tarefas complexas, composta pelo módulo *Long-Term Decision*.
- **Camada Executiva -** Responsável pelo sequenciamento do plano passado pela camada deliberativa, composta pelos módulos *Short-Term Decision* e *Motion*.
- **Camada Reativa -** Responsável por responder as percepções do ambiente, composta pelos módulos *Animation* e *Behavior*.

O fluxo de informações através dos módulos é indicado na Figura 1. Os dados captados do ambiente seguem para o módulo *Perception* que os filtra e os envia para os módulos responsáveis. *Perception*, após a filtragem, envia informações para *Location*, *Behavior*, *Animation*, *Short-Term Decision* e *Long-Term Decision*. Com a informação recebida de *Perception*, *Location* atualiza o modelo de ambiente do agente e disponibiliza, de forma síncrona com *Perception*, consultas a este modelo de ambiente para os módulos: *Long-Term Decision*, *Short-Term Decision* e *Motion*. Baseado no modelo atualizado do ambiente e nas percepções, *Long-Term Decision* utiliza sua representação interna de conhecimento para gerar planos que atendam aos objetivos do agente. Estes planos são passados para o *Short-Term Decision* que fica responsável pela execução dos mesmos. Para isto, *Short-Term Decision* utiliza as percepções e o modelo atualizado do ambiente juntamente com o módulo *Motion*, para decidir como executar cada tarefa de um dado plano. A execução das tarefas é feita trocando o comportamento reativo atual do módulo *Behavior*. Assim, *Behavior* atua sobre o ambiente baseado em suas percepções e o comportamento ativo atual. *Animation* e *Motion* são sub-módulos especializados de *Behavior* e *Short-Term Decision*, respectivamente. *Animation* comunica-se com o *Behavior* para decidir qual animação exibir e pode também receber percepções para gerar novas animações. Já o *Motion* é usado para decidir sobre a navegação do agente no ambiente.

3.1 Módulo Perception

Todos os dados captados por sensores do agente são tratados como percepções. Estes sensores podem ser dos mais variados tipos, imitando alguns dos sentidos humanos, sensores de visão, audição, tato e olfato. Cada percepção recebida deve ser repassada para os módulos apropriados. Assim, *Perception* centraliza o recebimento de percepções, cria um modelo de representação interna dessas percepções e filtra o envio dessas para cada módulo. Essa filtragem é importante para fazer que se chegue nos módulos apenas informações relevantes. Por exemplo, o som ambiente de que está ventando muito gera uma percepção auditiva que apesar de não ser muito útil para *Behavior*, pode ser interessante para *Long-Term Decision* concluir que vem chegando uma tempestade.

3.2 Módulo Location

O módulo *Location* possui uma representação do ambiente de modo a facilitar questões como determinação de rotas, navegação, desvio de obstáculos e localização de objetos no ambiente. Em jogos, é comum gerar essa representação de maneira pré-processada, diminuindo assim, o processamento em tempo de execução. Com isto, para o agente, *Location* atua como um grande oráculo sobre o ambiente, sendo atualizado diretamente por conjuntos de percepções.

3.3 Módulo Long-Term Decision

Ganhar o jogo é o mais importante objetivo para um BOT. Atingir esse objetivo significa tomar uma série de decisões que conduza o jogo ao estado de vitória para este BOT. *Long-Term Decision* é o módulo responsável pelas decisões de mais alto nível para o agente. Este módulo utiliza modelos de tomada de decisão para gerar soluções globais para tarefas complexas. A principal atribuição para este nível é o planejamento. Assim, *Long-Term Decision* fica responsável por decidir qual a meta atual e o melhor plano para atingir esta meta. Este plano é enviado para *Short-Term Decision*, que fica encarregado de executá-lo. Com isto, *Long-Term Decision* concentra-se em tipos de soluções estratégicas, delegando a execução dos planos para a camada inferior.

3.4 Módulo Short-Term Decision

Short-Term Decision atua como integrador das decisões tomadas pelo *Long-Term Decision* e as ações executadas pelo *Behavior*. Dado um plano passado para *Short-Term Decision*, este módulo fica responsável por decidir como executar o plano recebido. Este módulo também possui a liberdade de acrescentar ao plano, tarefas que otimizem o estado do agente e não desvie do objetivo do plano. A execução de tarefa dentro de um plano é feita selecionando os comportamentos dentro de *Behavior*. Dada uma seqüência de tarefas a ser executada, *Short-Term Decision* decide qual o comportamento ativar para a execução de cada uma destas tarefas. Uma tarefa pode conter um ou mais comportamentos seqüenciados. Dessa forma, *Short-Term Decision* atua sobre *Behavior* como um seletor do comportamento atual.

3.5 Módulo Motion

Motion é o responsável por entender como o agente deve se movimentar no mundo. Ele recebe informação de *Short-Term Decision* indicando para onde deve se movimentar e então decide a forma apropriada de se mover para o destino. Este módulo não executa a movimentação, apenas determina como executar. Toda decisão tomada por este módulo é computada com base nas estruturas de *Location*.

3.6 Módulo Animation

Animation é responsável por controlar o corpo do BOT decidindo qual animação será apresentada para demonstrar o estado atual de ação deste BOT. Estas animações são muitas vezes pré-geradas ou por animadores profissionais ou por captura de movimento, entretanto é possível gerar animação em tempo de execução[Grünvogel 2003]. Um exemplo de geração de animação em tempo de execução é a utilização de cinemática inversa[Eberly 2000].

3.7 Módulo Behavior

Behavior é o módulo responsável por executar a ação sobre o ambiente. Dado que *Short-Term Decision* selecionou um comportamento, este passa a ser o comportamento reativo atual. O comportamento reativo selecionado capta as percepções do ambiente e executa ações sobre o mesmo. O conjunto de ações do agente é similar às ações executadas pelo jogador humano através dos dispositivos de entrada.

4 Indigente Agent Framework

Nesta sessão, é apresentado o IAF - Indigente Agent Framework- um *framework* para o desenvolvimento de agentes cognitivos em jogos de primeira pessoa. Este *framework* fornece a estrutura para o desenvolvimento de agentes baseados na arquitetura apresentada em [Monteiro 2005]. O jogo Unreal Tournament 2004 foi escolhido como plataforma de estudo de caso devido a facilidade de interação com uma aplicação externa. A seguir é apresentada uma breve descrição do jogo e a organização do *framework* em questão.

4.1 Unreal Tournament 2004: O jogo

UT2004 é um jogo de tiro de primeira pessoa com suporte a multi-jogadores onde os personagens se enfrentam nos mais variados ambientes. Ele pode ser jogado de diversas maneiras dependendo da modalidade de jogo escolhida. Cada uma dessas modalidades oferece uma maneira distinta de jogar o mesmo jogo, mudando apenas as regras do mesmo.

As modalidades disponíveis em UT2004 são: Assault, Onslaught, DeathMatch, Capture The Flag, Team DeathMatch, Double Domination, Bombing Run, Mutant, Invasion, Last Man Standing e Instagib CTF. Dessas, duas se destacam e assumem o papel de modalidades clássicas, por estarem presentes em diversos outros jogos de tiro de primeira pessoa, são elas o DeathMatch e Capture The Flag. No formato do DeathMatch o jogador luta pela sua sobrevivência tentando ao máximo manter-se vivo enquanto destrói qualquer um que apareça em sua frente. Já no Capture The Flag, dois times de jogadores se confrontam com o objetivo de proteger a bandeira do time e sequestrar a bandeira inimiga.

Nas modalidades acima o jogo ocorre da mesma forma. O jogador tem a mesma visão que tem o personagem dentro do jogo(por isso a classificação como primeira pessoa), assim todas as percepções visuais do personagem são também as percepções dos jogadores. O personagem normalmente carrega uma arma a sua frente e tem conhecimento de seus pontos de vida e quão resistente está sua armadura. Como BOT também é um jogador, o agente modelado para implementar este BOT possuirá percepções similares a de um jogador humano.

Uma das principais vantagens de se utilizar o jogo UT2004 como laboratório é que ele é bastante personalizável. Novos módulos podem ser acrescidos utilizando uma linguagem própria que é o

Figura 2: *Tipos de Jogos Adicionados.*

Unreal Script. Devido a esta flexibilidade, surgiu , inicialmente no Instituto de Ciências da Informação da University of Southern California, o projeto *GameBOTs*[Andrew Scholer 2000][Kaminka et al. 2002] que será detalhado a seguir.

4.1.1 GameBots

O projeto *GameBOTs*[Andrew Scholer 2000][Kaminka et al. 2002] surgiu como uma iniciativa do Instituto de Ciências da Informação da University of Southern California em desenvolver uma modificação da primeira versão do jogos Unreal Tournament. A idéia deu tão certo que diversas universidades passaram a utilizar essas modificações e hoje já existem modificações para as versões mais recentes, como Unreal Tournament 2003 e UT 2004. Estas modificações funcionam como uma extensão das funcionalidades de controle do jogo, permitindo que personagens sejam controlados via socket TCP . Com isso, é possível desenvolver um agente que controla o personagem fazendo uso de qualquer linguagem de programação que tenha suporte a socket TCP.

Com a utilização da modificação, são criados quatro novos tipo de jogos. Estes novos são extensões de modos já existentes adicionando suporte a conexão via socket TCP. Os tipo são: Remote Team DeathMatch, Remote Capture The Flag, Remote Double Domination e Remote Bombing Run.

Quando o BOT está conectado ao jogo, informações sensoriais são enviadas pelo jogo através do socket de conexão e baseado nestas percepções o agente pode atuar enviando comandos através do mesmo socket. Os comandos enviandos definem como o personagem atua no ambiente, controlando movimentação, ataque e comunicação entre personagens.

Percepções e Ações A comunicação entre o agente e o jogo se dá através de troca de mensagens. O jogo envia mensagens síncronas e assíncronas com as informações sensoriais enquanto o agente atua também enviando mensagem. Uma descrição completa sobre as mensagens trocadas entre o agente e o jogo é apresentada em [Andrew Scholer 2000].

As mensagens síncronas chegam para o agente em lote num intervalo configurável. Elas incluem informações como a atualização visual do que o BOT vê e reporta o estado do próprio BOT. No início do lote das mensagens síncronas, o jogo envia uma mensagem de início do lote (BEG) e nessa mensagem contém o tempo ao qual o lote se refere. Logo após esta mensagem de início, todas as mensagens síncronas são enviadas até chegar uma mensagem de fim do lote (END). Todas as mensagens contidas no lote referem-se a um mesmo instante do tempo no jogo.

As mensagens assíncronas por outro lado, refletem os eventos ocorridos no jogo. Elas nunca aparecem entre um BEG e um END já que elas não são síncronas. Estas mensagens representam as coisas que podem acontecer a qualquer momento aleatório do jogo, tais

como: tomar dano; difusão de mensagem por algum outro personagem; bater na parede. Sempre que ocorrer um evento no jogo que gere uma mensagem assíncrona, esta mensagem estará antes ou depois de um lote de mensagem síncrona. Entretanto, não é possível garantir que a mensagem assíncrona refere-se ao mesmo tempo do jogo que o lote anterior ou posterior.

As ações dos BOTs são determinadas pelas mensagens enviadas pelo programa do agente. Essas mensagens seguem o mesmo estilo de formatação das percepções que chegam, que é um identificador seguido de zero ou mais argumentos. A maioria dos comandos possuem efeitos persistentes, o que significa que comandos como movimento e rotação, uma vez iniciados, só irão parar quando alcançar o destino, e o comando de atacar mantém o personagem atirando até que seja enviado um outro comando de parada do ataque.

No *framework*, foi desenvolvido uma camada de abstração para mensagens sensoriais e os comandos. Para toda mensagem que chega no agente é feito o parser de cada uma e instanciados objetos de mensagens que contém as informações da mensagem de fácil acesso para o *framework*. Quanto aos comandos, uma classe chamada MailBox se encarrega de enviar cada comando possível para o jogo através de chamadas de métodos. Esta mesma classe MailBox, fica responsável por receber as mensagens e encaminhá-las para que seja feito o parser.

4.2 Organização do IAF

O IAF é feito essencialmente em C++, utilizando os principais conceitos de orientação a objeto. Ele é separado em pacotes onde são agrupadas as diversas funcionalidades para dar suporte a criação de um BOT.

Os pacotes do *framework* fornecem as mais variadas funcionalidades. Estas funcionalidades estão agrupadas basicamente em: rede, arquitetura, matemática e máquina de estados finitos. A seguir, são descritos os conteúdos de cada pacote juntamente com o funcionamento dos mesmos

4.2.1 Comunicação

A comunição entre agente e jogo se dá através de troca de mensagens. Devido a isto foi necessário o desenvolvimento de todo o suporte para comunição em rede. Neste suporte está incluido desde a conexão através socket, até a transformação da mensagem em um tipo abstrado de dado[3]. Desta forma, as mensagens passam a ser utilizadas pelo *framework* como informação e não mais como dados.

Uma classe MailBox é responsável por receber e enviar as mensagens. Como a quantidade de comandos é pequena, o envio deles se dá através de chamadas de método da classe MailBox. Já para o recebimento das mensagens, é utilizado uma thread com espera bloqueante que *bufferiza* os pacotes chegados formando mensagens que são enviadas para a fábrica de mensagens sensoriais.

A fábrica de mensagens sensoriais, o SensoryMessageFactory, é responsável por, dado uma mensagem em texto plano, no formato TYPE Arg1 Values1 ... ArgN ValuesN, gerar informação útil para o *framework*, ou seja, criar uma instância referente ao tipo de mensagem, com as operações necessárias para o acesso da mesma. Para isso, a fábrica realiza o parser das mensagens e decide qual instância de mensagen sensorial deve criar. As mensagens criadas são todas filhas de SensoryMessage, e elas se dividem em dois tipos básicos que são as mensagens síncronas e assíncronas.

As mensagens síncronas são recebidas pelo agente em um intervalo configurável e informam essencialmente o que o BOT vê, o estado do mesmo e a pontuação no jogo. Já as mensagens assíncronas informam eventos que podem ocorrer a qualquer instante no jogo, como: ser atingido, comunicação textual, mudança de ambiente, colisão com alguma parede ou jogador, dano tomado, entre outros.

A entrega dessas mensagens é feita pelo próprio MailBox, que mantém uma lista de inscritos interessados em mensagens chega-

[3]Tipo usado para encapsular outros tipos de dados e que possuem operações associadas

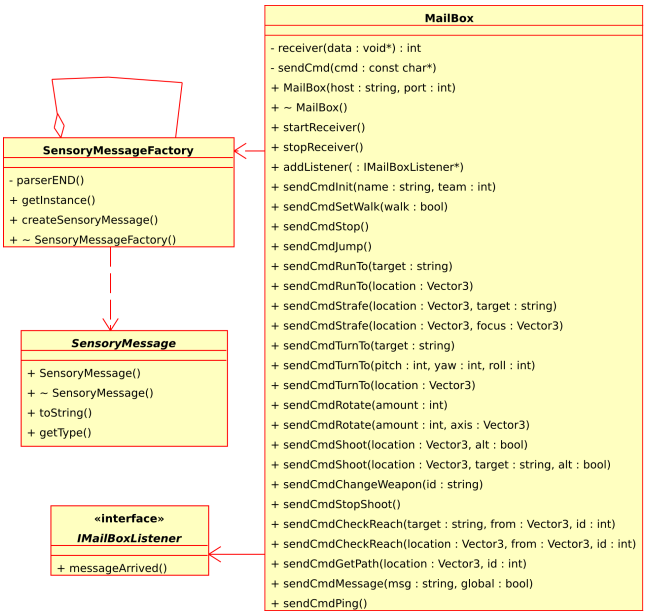


Figura 3: *Conjunto de classes e interface utilizada na comunicação.*

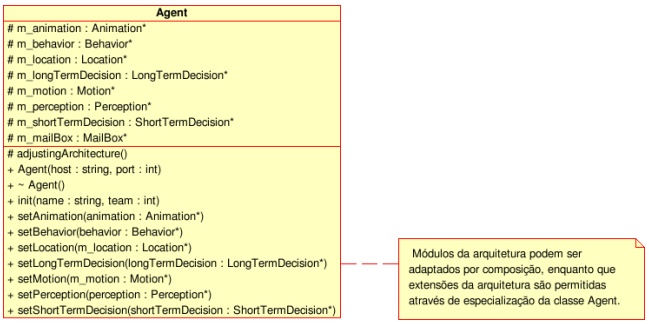


Figura 4: *Classe básica para a composição de um agente.*

das. Estes interessados precisam implementar IMailBoxListener para então se inscrever. Uma vez inscrito no MailBox, para toda mensagem chegada, será gerada um SensoryMessage para ele.

Utilização da Comunicação: A utilização dos componentes de comunição do *framework* permeia o uso da classe MailBox, que é utilizada tanto para o envio como o para o recebimento de mensagens. Novas mensagens podem ser criadas com a especialização da classe SensoryMessage, como pode ser visto na Figura 3.

4.2.2 Implementação da Arquitetura

O pacote architecture fornece a implementação da estrutura proposta pela arquitetura definida em [Monteiro 2005]. Neste pacote, são organizados cada um dos módulos da arquitetura juntamente com seus relacionamentos. Uma importante classe deste pacote é a classe Agent que contém a estrutura de um agente cognitivo que implementa a arquitetura proposta. Esta classe Agent é de fácil extensão o que permite sua reutilização também para evoluções que venham ocorrer na arquitetura. Ela implementa todas as funcionalidades necessárias para se criar um BOT, o que inclui toda a parte de comunicação com o jogo. A seguir são descritos os módulos implementados.

Utilização da Arquitetura: A classe Agent é a classe-base para o desenvolvimento de um agente no IAF . Ela define todos módulos da arquitetura e sua adaptação é feita através da composição de novos módulos, como pode ser visto na Figura 4.

Perception O módulo *Perception* tem como principal funcionalidade agrupar as informações sensoriais e entregar para os devidos módulos as informações necessárias. Ele implementa IMailBox-

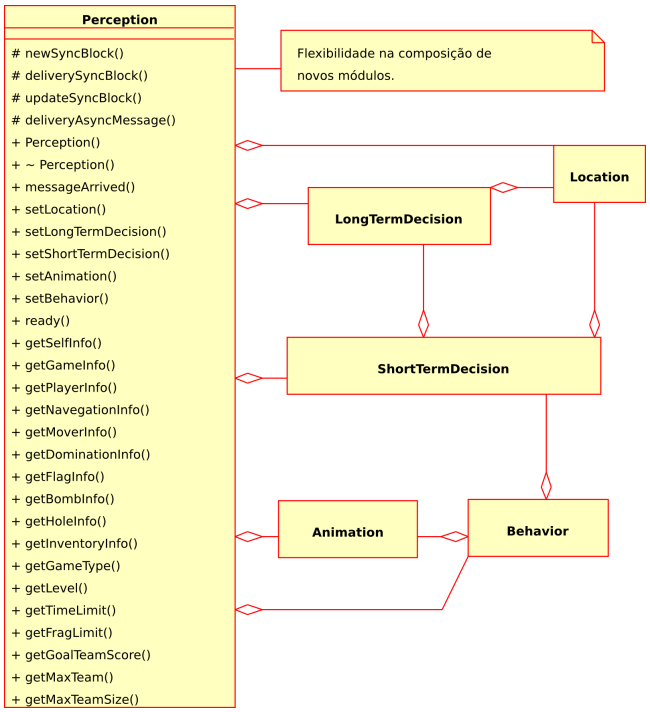


Figura 5: *O módulo perception como proxy das percepções do agente*

Listener, o que significa que ele é notificado pelo MailBox sempre que um nova mensagem chega.

Como as informações sensoriais podem ser síncronas ou assíncronas, *Perception* dá um tratamento diferenciado para cada um dos dois tipos.

As mensagens assíncronas são enviadas contendo as informações do evento que disparou, desta forma, cada evento contém a sua mensagem assíncrona equivalente. Já para as mensagens síncronas, o que as disparam é o temporizador, mas neste caso o temporizador não dispara apenas uma mensagem contendo a informação necessária, mas sim, um conjunto de mensagens delimitado por mensagens de BEGIN e END. O módulo *Perception* é quem se encarrega de agrupar essas mensagens síncronas e só entregar para os outros módulos o bloco completo após o recebimento do END. Além disso, é possível fazer a seleção de quais mensagens sensorias devem ser passadas como percepção para cada módulo.

Uma outra funcionalidade do *Perception* é manter o histórico centralizado de percepções. Como a arquitetura é bem modular, evita que cada módulo tente manter esse histórico a fim de realizar planejamento ou tomadas de decisões. As mensagens são armazenadas em uma fila e o tamanho dessa fila é configurável.

Utilização de Perception: Todas as percepções passam pela classe *Perception* através de troca de mensagens.O método messageArrived define o que deve ser feito para cada mensagem recebida. Neste método é que são separadas as mensagens síncronas de assíncronas para entrega no demais módulos. A estrutura de *Perception* é apresentada na Figura 5.

Location O módulo *Location* utiliza a representação interna do ambiente através de *WaypointSystem* e ele pode ser facilmente extendido para utilizar mais alguns outros tipos de representação, como campos potenciais[Tozour 2004] ou mapas de influências[Schwab 2004].

O *WaypointSystem* é um grafo não orientado que é atualizado inserindo-se um novo nó a cada novo marco encontrado no jogo, conforme a figura 9. Sobre este grafo são permitidos, além das operações de construção do grafo, a consulta sobre o caminho mínimo entre os dois marcos e a verifação de nós já visitados.

Location tem importante papel no processo cognitivo, servindo

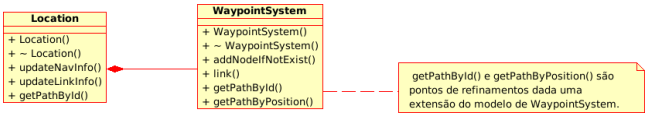


Figura 6: *Location utilizando po padrão o WaypointSystem.*

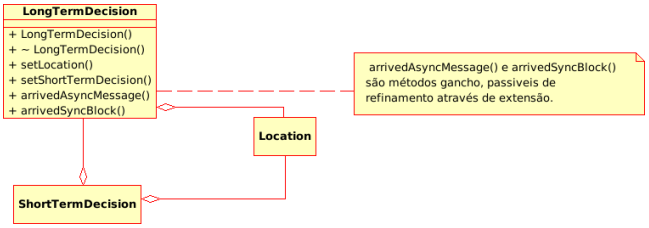


Figura 7: *Os relacionamentos de LongTermDecision e seus hots-pots.*

como uma representação interna do ambiente. O auxílio no planejamento de trajetória é apenas umas das importantes funcionalidades fornecidas por esse módulo. Informações sobre os nós visitados e as características desses nós e suas vizinhanças auxiliam bastante no processo decisório do agente.

Utilização de Location: Cada nova mensagem de navegação é tratada pelo método *updateNavInfo*, que por padrão atualiza o *WaypointSystem*. Já para as mensagens de informação de caminho que chegam, o método *updateLinkInfo* é chamado. Na implementação padrão, este último método atualiza as ligações entre os nós do *WaypointSystem*. O método *getPathById* é também outro ponto de flexibilidade permitindo que especializações de *Location* definam a forma como o caminho no ambiente é gerado. A estrutura de *Location* pode ser vista na Figura 6.

Long-Term Decision *Long-Term Decision* é o módulo utilizado para implementar a camada deliberativa do agente. Sua implementação no *framework* é apenas estrutural, deixando a flexibilidade de extensão para a fase de projeto do agente. Sua estrutura facilita integração com outras arquiteturas baseadas em conhecimencimento como o SOAR[Laird et al. 1987]. A idéia principal desse módulo é gerar planos que devem ser executados pelo *Short-Term Decision*.

Utilização de LongTermDecision: Permite a especialização de uma camada deliberativa para o agente através dos métodos gancho *arrivedAsyncMessage* e *arrivedSyncBlock* , como pode ser visto na Figura 7. Os métodos *arrivedAsyncMessage* e *arrivedSyncBlock* são chamados por *Perception* com a chegada das mensagens sensoriais.

Short-Term Decision O módulo *Short-Term Decision* é o responsável pelo seqüenciamento das reações do agente, gerando um comportamento complexo de alto nível. Sua implementação no *framework* é apenas estrutural, deixando em aberto a forma como o seqüenciamento dos comportamentos é feito. Uma forma simples da sua implementação é utilizando máquinas de estado finito, entretanto, neste aspecto as máquinas possuem um grande inconviniente que é o da previsibilidade das transições de estado.

Utilização de ShortTermDecision: Os métodos *arrivedAsyncMessage* e *arrivedSyncBlock* são métodos gancho, ou seja, permitem a adaptação de *ShortTermDecision* através de sua sobrescrita numa especialização. A Figura 8 apresenta a estrutura de *ShortTermDecision*.

Motion O módulo *Motion* é utilizado principalmente para o planejamento de trajetória. Apesar do jogo UT2004 já fornecer suporte interno para o planejamento de trajetória, este suporte é limitado no que se refere a interferência no processo de planejamento. Com ele, não é possível, por exemplo, escolher o caminho com melhor utilidade ao invés do menor caminho. Essa deficiência é superada

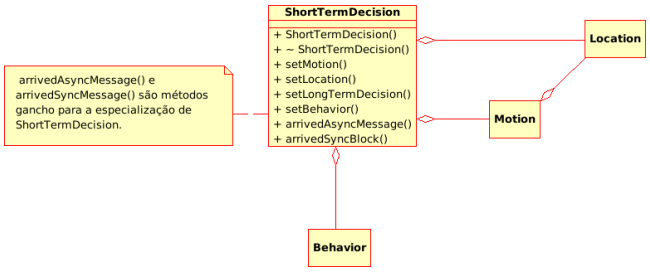


Figura 8: *ShortTermDecision e seus relacionamentos.*

Figura 9: *Marcos visualizados pelo BOT, representado como cilindros preto quando habilitado o modo de debug.*

devido a criação do modelo interno do ambiente, que é a fonte de dados para o módulo *Motion*.

Motion implementa essencialmente o algorimo A* sobre a representação em grafos de *Location*. Entretanto, com a extensão de *Motion*, é possível gerar novas formas de planejamento de trajetória, o que não é permitido com a representação fixa oferecida pelo jogo.

Utilização de Motion: *Motion* oferece o método *getPath* que pode ser sobrescrito para atender a novos modelos de ambiente. A Figura 10 apresenta a estrutura de *Motion*.

Animation O módulo *Animation* possui apenas sua implementação estrutural e um dos principais motivos para isto é que o jogo UT2004 não fornece uma maneira direta de controlar a animação do personagem. Esta animação fica a cargo do prórpio jogo. Mesmo assim, *Animation* pode ser extendida para suportar os diversos modos de se animar um personagem. Como é bastante comum a utilização da técnica de keyframe para animação, uma extensão de *Animation* junto a uma máquina de estado seria uma alternativa imediata.

Utilização de Animation: O controle da animação pode ser definido pelas mensagens que chegam através do *arrivedAsyncMessage* e *arrivedSyncBlock*. A Figura 11 apresenta a estrutura de *Animation*.

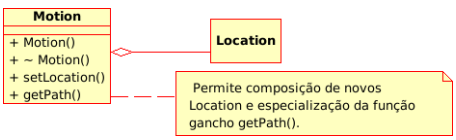


Figura 10: *Modulo de acesso à entidade Location.*

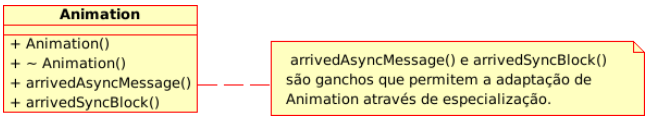


Figura 11: *Implementação estrutural do módulo responsável pela animação dos BOTs.*

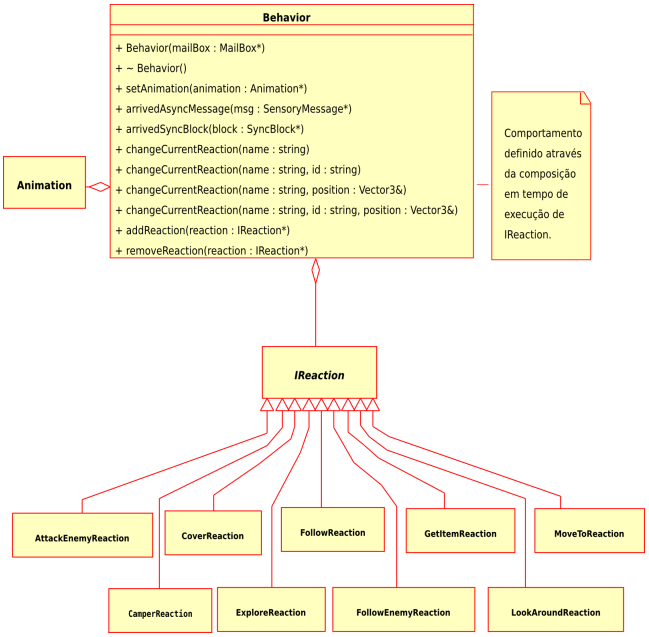


Figura 12: *Modulo behavior com adaptações através de composição.*

Behavior O módulo *Behavior* atua diretamente com o ambiente, enviando os comandos básicos definidos pelo jogo. Ele possui uma lista de comportamentos na qual apenas um é ativado por vez. Estes comportamentos são ativados pelo módulo *Short-Term Decision*, e uma vez ativado, ele passa a responder por todas as percepções que chegem a *Behavior*. Apesar da possibilidade de implementação dos comportamentos de diferentes formas, o *framework* pré-disponibiliza alguns comportamentos implementados como máquinas de estados finitos. Esses comportamentos são utilizados como reações elementares pelo *Short-Term Decision*, que forma comportamentos complexo através de composição.

Os comportamentos implementados por padrão são específicos para a atuação nos jogos UT, UT2003 e UT2004. Uma importante observação quanto as máquinas de estados é que elas podem compartilhar estados com outras máquinas. Desta forma, um comportamento de ”Tocaia”que necessite de um estado de ”Aguardando Inimigo”, pode compartilhar esse mesmo estado com um outro comportamento de ”Protegendo Item”, diminuindo o esforço com a implementação.

Utilização de Behavior: O funcionamento de *Behavior* é dependente das *IReaction* que o compõe. As *IReaction* são selecionadas como o comportamento ativo através do método *changeCurrentReaction*. Desta maneira, os métodos *arrivedASyncMessage* e *arrivedSynBlock* são repassados para o recebimento de mensagens sensoriais da *IReaction* ativa. Novos comportamentos podem ser definidos através da especialização de *IReaction* que também disponibiliza os *hotspots arrivedAsyncMessage* e *arrivedSyncBlock*. A Figura 12 apresenta a estrutura de *Behavior*.

4.2.3 Funcionalidades Matemáticas

Funcionalidades referentes a matemática são necessárias em diversos momentos da implementação de um BOT, principalmente as funcionalidade referentes ao cálculo vetorial. Isto porque, constantemente o BOT necessita calcular qual sua distância em relação a algum ponto, a nova posição de um determinato objeto, ou até quais

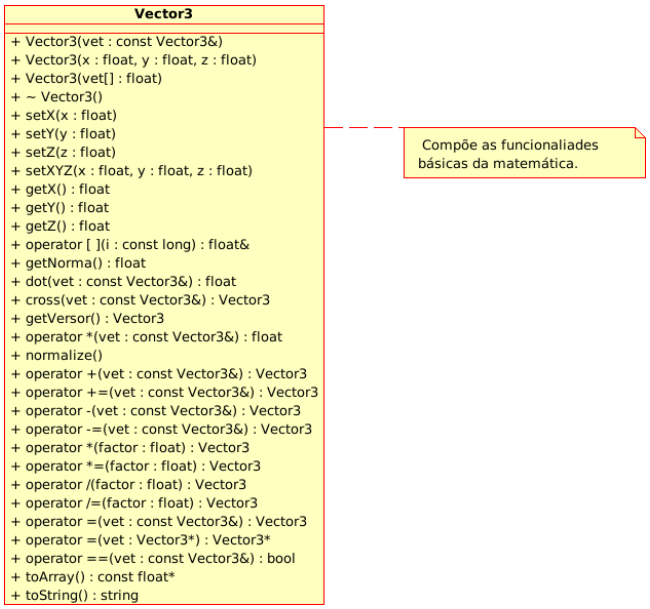


Figura 13: *Funcionalidades do Cálculo Vetorial.*

direções é possível se movimentar para desviar de um projétil a caminho. No intuito de suprir esse importante papel, está presente no pacote *Math* a classe *Vector3*.

Vector3 é uma classe que agrega grande parte das operações possíveis no cálculo vetorial para um vetor de três dimensões. Além das operações básicas de soma e adição de vetores, é permitido também multiplicação e divisão por escalar, produto vetorial, normalização, obtenção da norma e do versor. Tudo isso com o máximo de sobrecarga de operadores, tornando mais natural a utilização desse tipo.

Utilização de Math: As funcionalidades básicas do cálculo vetorial são disponibilizadas na classe Vector3, como mostrado na figura 13.

4.2.4 Máquina de Estados Finitos

A utilização de máquinas de estados finitos é a forma mais difundida de implementação de comportamentos para BOT em jogos de primeira pessoa[Schwab 2004]. Como citado anteriormente, suas desvantagens podem ser superadas pelo uso racional deste recurso. Aliado a isto, a implementenção no *framework* da estrutura para máquinas de estados é bastante flexível, explorando bem os recursos de orientação a objetos.

A classe *FSMState* é uma classe básica para qualquer estado que venha ser inserido na máquina de estado. Assim, a criação de um estado se dá através da herança de *FSMState*. Os métodos de *FSMState* são chamados pela máquina que contém a classe, de maneira que para implementar um novo estado basta codificar o que faz cada evento. É lançado um evento sempre que um estado é inicializado, quando se entra nele e também quando se sai dele. Um método de *update* é chamado sempre que uma nova entrada chega para o *FSMState*.

A classe *FSMMachine* representa a máquina de estado. Ela tem uma interface simples e é facilmente extendida para se criar um comportamento. Para a implementação de um novo comportamento basta determinar a função de transição de estados, pois todo o restante já é implementado em *FSMMachine*.

Um ponto importante da implementação de máquina de estados finitos no *framework* é que ela suporta máquina de estados hierárquica. Ou seja, uma máquina de estado pode conter estados que também são máquinas de estado. Esta abordagem reduz bastante no número de transições entre os estados e facilita o entendimento da modelagem para um comportamento complexo[dos Santos 2004].

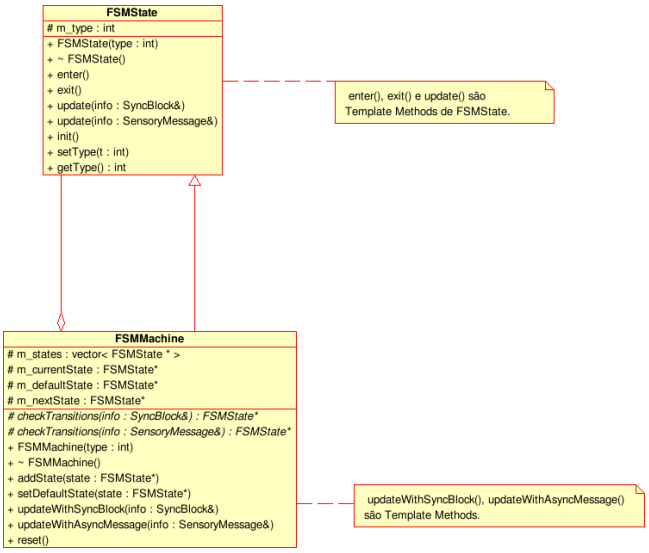


Figura 14: *Estrutura para o desenvolvimento de máquinas de estado.*

Utilização da Máquina de Estados: A Classe *FSMMachine* oference as operações definidas para funcionamento de uma máquina de estado e mantém dois *hotspots*, *updateWithSyncBlock* e *updateWithAsyncMessage*, que avaliam a transição de estados. Ela é uma especialização de *FSMState*, o que a permite atuar como estado numa máquina de estado hierárquica. A classe *FSMState* também tem o papel de Template Methods e disponibiliza os *hotspots*, *enter*, *exit* e *update*, chamados respectivamente na entrada do estado, na saída para outro estado e na atualização do estado. A Figura 14 apresenta a estrutura das máquinas de estados.

5 Estudo de Caso

Para avaliar a instanciação de BOTs utilizando o IAF , dois cenários foram montados. O primeiro envolveu o desenvolvimento de um BOT puramente reativo, ou seja, utilizando apenas um comportamento e sem fazer uso do *Short-Term Decision* e do *Long-Term Decision*. O segundo cenário foi a implementação de um agente cognitivo simples, que possuia uma meta padrão e uma representação do seu ambiente. Nestes dois casos, foi possível perceber não só a simplicidade de implementação dos BOTs, com poucos *hotspots* para serem especializados, como também a eficiência obtida pelos jogadores virtuais nos combates.

É importante caracterizar o ambiente no qual o agente é inserido. Isto porque a maneira como o ambiente é classificado influi diretamente na forma como o agente é projetado e também possibilita uma melhor compreensão das dificuldade envolvidas no processo de desenvolvimento do BOT. O jogo apresenta o ambiente com as seguintes características: **parcialmente observável**, porque o agente, através de suas percepções, não consegue acessar todos os estados relevantes ao ambiente; **estocástico**, porque o estado seguinte do ambiente não depende exclusivamente das ações do agente; **seqüencial**, porque a qualidade das ações do BOT no estado atual depende das decisões nos estados anteriores; **dinâmico**, porque entre o agente perceber e agir o ambiente já mudou de estado; **contínuo**, porque existe um número ilimitado e não enumerável[4] de como as percepções são apresentadas; **multiagente**, porque existe tanto interação cooperativa como competitiva entre os agentes[Russel and Norving 2004].

O estilo de jogo Team DeathMatch foi utilizado para simplificar

[4]Apesar de toda representação computacional ser recursivamente enumerável, a informação que este modelo computacional respresenta não é enumerável. Um exemplo é que as posições dos objetos são passadas em coordenadas cartesianas com cada componente pertencente ao conjunto dos reais. Apesar do modelo computacional suportar uma representação parcial dos reais, o conjunto dos números reais são não enumeráveis. Por isso é possível dizer que as percepções são não enumeráveis.

os testes. Neste estilo, dois grupos travam combate e vence o grupo que primeiro alcançar a pontuação de objetivo da partida. A pontuação é contada com o número de baixas do inimigo. Cada personagem possui sua pontuação própria que se soma a dos outros integrantes para formar a pontuação da equipe. A avaliação dos BOTs é baseada exclusivamente em sua pontuação dentro da partida.

Nos dois cenários foram utilizados as seguintes configurações da partida:

- **GameType:** Remote Bot Team DeathMatch
- **Bot Skill:** Skilled
- **Goal Score:** 50 no primeiro ensaio e 100 no segundo
- **Time Limit:** 40
- **Map:** DM-Rustatorium

A única diferença existente na configuração da partida para o primeiro e o segundo cenário foi a pontuação de objetivo, que no primeiro foi 50 e no segundo foi 100. A habilidade dos demais BOTs estava configurada como ”Skilled”que é um nível a mais da habilidade que vem configurada como padrão. As habilidades para os BOTs do jogo disponíveis são: Novice, Average, Experienced, Skilled, Adept, Masterful, Inhuman e Godlike. O mapa escolhido foi o DM-Rustatorium por ser um mapa amplo e de fácil navegação, o que evitaria que o BOT reativo ficasse preso em um mínimo local[5] por não possuir uma representação do ambiente.

No primeiro cenário, para implementação do BOT foi utilizado apenas um dos comportamentos padrões, o de exploração. O comportamento de exploração já é definido como o inicial dentro do módulo *Behavior*, o que significa que a instanciação da classe Agent já cria um agente reativo pronto para responder aos estímulos do jogo. Este agente, por não possuir cognição associada, é considerado incompleto aos propósitos do IAF , que é o desenvolvimento de agente cognitivo. Para alcançar tal objetivo são necessárias especializações de alguns módulos da arquitetura. O agente reativo implementado, chamado de *SimpleExplorer*, apenas percorria o mapa buscando os marcos existentes no ambiente e combatendo seus adversários. Em uma partida real de jogo, a classificação final foi a seguinte:

Time A		**Time B**	
BOTs	Pontuações	BOTs	Pontuações
SimpleExplorer	**14**	Cannonball	12
Tamika	14	Kaela	10
Prism	13	Arclite	8
Jakob	9	Subversa	3
-	-	Gorge	4

A participação dos demais BOTs na partida foi aleatória. O que significa que não houve seleção do grupo que iria compor o time do *SimpleExplorer*. Mesmo com um grupo em desvantagem numérica, ele obteve um resultado muito bom comparado aos BOTs originais do jogo, superando inclusive o esperado para um BOT puramente reativo.

Já para o desenvolvimento do segundo agente, o *Explorer*, foi necessário especializar apenas uma classe, a *ShortTermDecision*, sobrescrevendo os métodos-gancho *arrivedAsyncMessage* e *arrivedSyncBlock*. Essa classe especializada passou a fazer parte da instância de Agent através de composição. Dado que os comportamentos padrões já são implementados. Neste segundo cenário, o agente já fazia uso de sua representação interna. Ele decidia se seguiria um marco, se perseguiria um personagem ou se pegaria um item ao chão. Em sua partida, o BOT teve o seguinte resultado:

[5]O agente reativo escolhe seu próximo nó avaliando apenas suas percepções, então numa tentativa de explorar o mapa, a sua escolha pode ser expressa como um função de avaliação onde sua percepção só lhe permite achar a sua melhor escolha local, se sua percepção não lhe permitir visualizar além de suas opções locais, é possível que a escolha da melhor opção se repita indefinidamente. Desta forma dizemos que o agente ficou preso em um mínimo local.

Time A		**Time B**	
BOTs	Pontuações	BOTs	Pontuações
Explorer	**46**	Ambrosia	30
Skakauk	24	Enigma	22
Guardian	20	Remus	15
Satin	10	Reinha	11
-	-	Vírus	5

Apesar do foco deste trabalho ser o desenvolvimento do *framework* e não dos BOTs, a instanciação desses BOTs ajuda a consolidar a utilidade do *framework* para a criação desses jogadores virtuais. É possível perceber com estas implementações é que a simplificação no processo de desenvolvimento não implica em uma baixa eficiência do agente. Os resultados ajudam a reforçar isso, e uma explicação plausível para tal é que é possível concentrar esforços no que o agente irá fazer ao invés de como ele irá fazer.

6 Considerações Finais

A utilização de jogos eletrônicos como laboratório para testar tanto teorias sobre o raciocínio humano quanto métodos de resolução de problemas vem crescendo ao longo do tempo. Os desafios apresentados pelos jogos eletrônicos atuais requerem um grande esforço para o desenvolvimento de personagens virtuais. Devido a isto, a produção de ferramentas que colaborem com este processo ganha cada vez mais apoio tanto do meio acadêmico quanto da indústria de jogos.

A arquitetura apresentada aqui é baseada no modelo híbrido de três camadas, o que oferece melhor aproximação com as seqüências de ações dos humanos e com o seu raciocínio. A divisão da arquitetura em sete módulos quebra o processo de desenvolvimento do agente em partes menores, simplificando o gerenciamento de cada parte e reduzindo a complexidade total do problema. Além disso, a especialização no contexto de jogos é o que permite que esta arquitetura de agente cognitivo obtenha melhores resultados em relação a outras arquiteturas híbridas.

A utilização de uma estrutura definida, aliado a uma arquitetura de agente, servindo de suporte para a organização e o desenvolvimento de BOTs, simplifica bastante a criação desses jogadores virtuais. Neste trabalho, a estrutura fornecida através do Indigente Agent Frameworkcumpre com o objetivo de simplificação do processo de desenvolvimento de agentes cognitivos para jogos. Isto porque, o IAF utiliza as vantagens apresentadas pela arquitetura de agente, disponibiliza diversas funcionalidades necessárias para a criação de BOTs, e fornece um estrutura extensível para a organização dessas funcionalidades.

A avaliação dos resultados, tanto da atuação quanto do desenvolvimento, permite observar que além de simplificar o processo de desenvolvimento de agentes, é possível obter facilmente bons resultados devido ao nível de abstração oferecido pela arquitetura. Com isto, é dado mais um passo na busca pela criação de jogadores virtuais convincentes.

6.1 Trabalhos Futuros

Apesar dos grandes avanços conseguidos com o IAF , algumas atividades ainda podem ser desenvolvidas no intuito de aperfeiçoá-lo. A implementação do suporte a utilização de linguagem de script como LUA[Ierusalimschy 2006] é uma delas. O suporte a linguagem de script permite que o *framework* seja reconfigurado sem a necessidade de recompilar seu código fonte. Esta configuração vai desde parametrização de atributos iniciais, passando por redefinição dos comportamentos, até a especialização de diversas estruturas presentes no *framework*.

Extensões da arquitetura serão os passos seguintes, adequando esta a novos contextos. Há em vista uma extensão para viabilizar o planejamento multiagente, pois hoje isto fica por conta da mesma estrutura responsável pelo planejamento local. Extensões da arquitetura refletirão em incremento do *framework*. Com isto a intenção é que o IAF passe a dá suporte ao desenvolvimento de agentes cognitivos especializados para cada novo domínio.

Referências

ADOBBATI, R., MARSHALL, A. N., SCHOLER, A., TEJADA, S., KAMINKA, G. A., SCHAFFER, S., AND SOLLITTO, C. 2001. Gamebots: a 3d virtual world test-bed for multi-agent research. *Proceedings of 2nd International Workshop on Infrastructure, MAS and MAS Scalability*.

ANDREW SCHOLER, G. K., 2000. Gamebots. Último acesso em 01 de dezembro de 2006.

DE BYL, P. B. 2004. *Programming Believable Characters for Computer Games*. Charles River Media.

DOS SANTOS, G. L. 2004. *Máquinas de Estados Hierárquicas em Jogos Eletrônicos*. Master's thesis, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro.

EBERLY, D. H. 2000. *3D Game Engine Design*. Morgan Kaufmann.

GRÜNVOGEL, S. M. 2003. Dynamic character animation. *International Journal of Intelligent Games and Simulation. 2*, 1, 11–19.

IERUSALIMSCHY, R. 2006. *Programming in Lua*. Lua.Org; 2nd edition.

KAMINKA, G. A., VELOSO, M. M., SCHAFFER, S., SOLLITTO, C., ADOBBATI, R., MARSHALL, A. N., SCHOLER, A., AND TEJADA, S. 2002. Gamebots: a flexible test bed for multiagent team research. *Commun. ACM 45*, 1, 43–45.

LAIRD, J. E., NEWELL, A., AND ROSENBLOOM, P. S. 1987. Soar: an architecture for general intelligence. *Artif. Intell. 33*, 1, 1–64.

MONTEIRO, I. M. 2005. Uma arquitetura modular para o desenvolvimento de agentes cognitivos em jogos de primeira e terceira pessoa. In *Anais do IV Workshop Brasileiro de Jogos e Entretenimento Digital*, 219–229.

NILSSON, N. J. 1998. *Artificial Intelligence - A New Synthesis*. Morgan Kaufmann.

RUSSEL, S., AND NORVING, P. 2004. *Inteligência Artificial - Tradução da segunda edição.* Editora Campus.

SCHWAB, B. 2004. *AI Game Engine Programming*. Charles River Media.

TOZOUR, P. 2004. Search space representations. In *AI Game Programming Wisdom 2*, Charles River Media, 85–102.

VAN WAVEREN, J. M. P. 2001. *The Quake III Arena Bot*. Master's thesis, Delft University of Technology, Delft, Netherlands.

Interperceptive games

Julio César Melo
Rummenigge R. Dantas
Luiz Marcos G. Gonçalves
DCA - UFRN

Claudio A. Schneider
Josivan Xavier
Samuel Azevedo
DIMAP - UFRN

Aquiles Burlamaqui
UERN

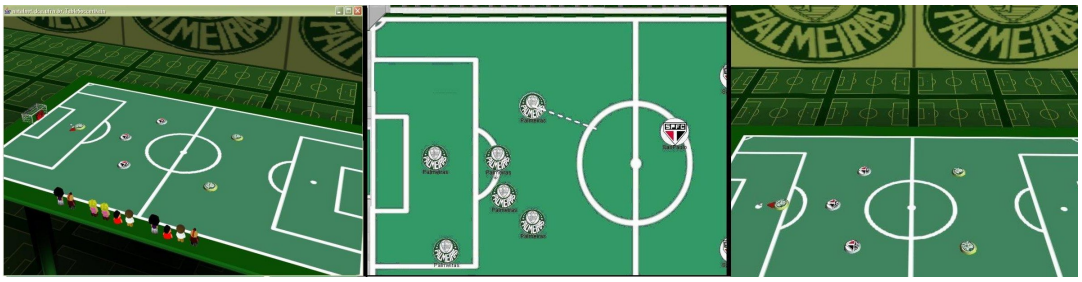


Figure 1: Interperceptive game screeshots

Abstract

In this work, we explain how the construction of inter-perceptive games is becoming a tendency, instead of conventional ones. We propose a manner for performing inter-perceptive games, showing the main characteristics of the architecture. The huge number of users, aimed by these applications, can make the difference between success and failure for such games. Thus, a case study using a table soccer game (or also called button soccer) is presented as an inter-perceptive game for a large quantity of players. Experiments show the applicability of this architecture to other more general games.

Keywords:: Interperception, Multi-User games

Author's Contact:

{julio.engcomp, rudsondant, claudio.schneider, josivanx, samuel.azevedo aquilesburlamaqui}@gmail.com
lmarcos@dca.ufrn.br

1 Introduction

With the evolution of game industry, many console models and platforms have appeared with different characteristics and also based upon different hardware and software resources. In relation to their actual graphical processing capacity, the consoles started from two-dimensional graphical interfaces through isometric maps which introduced a three-dimensional depth notion on the screen, and finally achieving the interfaces with three-dimensional graphics. We can illustrate the games evolution by the variety of hardware devices developed, such as light guns for shooting games, steering wheel or paddles for racing games, and dancing pads, among others.

In order to improved theirs marks on the market, console developers have to produce better devices than the existing ones. These new devices improve characteristics as the graphical capacity and types of interaction. Consequently many consoles generation appeared. These generations are normally labeled by the word size of their processors, thus we have consoles with 8 bits, 16 bits, and so on. An interesting aspect of this situation is that notwithstanding the rise of new games and consoles, they coexist in the market with the old ones. And some game companies continue producing new games for old platforms. This strategy allows these companies to reach more consumers using their products.

Nowadays, it is common to see people using not just game consoles, but also Personal Computers, cellular phones, personal digital assistants (PDAs), tablets, and mobiles devices, between other hardware devices, to play games. The rise and spread of Internet for these electronic devices have allowed the development of on-line games. It is known that the number of multi-user on-line games(MOG) players is increasing and that there is a huge sort of this kind of games (shotting, racing, RPG, etc), each one with its own requirements to execute.

Besides, the evolution in terms of hardware, software, and network, related to multi-user games did not stop the production and the demand for old-style games, and there are lots of users still playing games that had appeared decades ago. An example of this situation is the MUDs(MultiUser Dungeons) [Bartle 2007], a game that consists of multi-user virtual environments placed into a medieval scenery built in a textual interface. In a similar way, on the other hand we also have millions of users using the newest created games.

Why do some users prefer to continue using old technology instead new ones? In relation to on-line games, the differences between interaction devices, size of processor's word and network capacities are some of the factors that limit the choice for a game against others. Users using computers with a low processing power and a dial up connection would prefer simpler games that are easier to execute by their hardware, while users using the last hardware's generation with a broadband connection would prefer games with more advanced interfaces.

All of these differences became a problem because they create a barrier between players, isolating them into different groups. In this manner, players that by affinity could be part of the same group may be excluded from that group for more complex applications by the lack of hardware/software requirements.

In order to solve this problem, we apply the inter-perception concept [Azevedo et al. 2006] to games, allowing users with different computational resources to participate (through different user interfaces) of a same shared game environment, creating the concept of an Inter-perceptive Game.

2 Related Work

In order to better understand inter-perception, we discuss about some works which also have as objective the integration of players with different devices. The Cross-Media Games [Lindt et al. 2005] are focused on the large variety of game consoles, mobility and other devices that can be used together to allow a new game experience. That work presents a study about games where the players have different modes to play, that are provided by different devices. The players can communicate with each other by SMS, but they do not share the same game environment, which is the focus of our approach.

The Cross-platform [Han et al. 2005] is an engine which is proposed to develop 3D games accessible to different platforms. In their article they present an architecture for multi-user games where the player can access the game from these different devices and share the same game environment. This architecture uses a single

server to provide to the players the capability of interacting in the same game environment, in spite of their console. However, this work only focuses on games with 3D interfaces while our approach abstracts the interface used in the game.

RuneScape [West 2007] is a massively multiplayer on-line role-playing game (MMORPG) and the name of a medieval-like virtual world where players can role characters with many different skills, improoving the choosen skills without the limitations of a "character class". The players of this game may choose one between two interfaces which are different in terms of theirs graphical quality, but booth are three-dimensional interfaces of the RuneScape virtual environment. So, in spite of having two different interfaces for the same environment, it is an example of a not truly interperceptive game because the messages used within it don't need to be translated between the interfaces.

Amphibian [Bittencourt et al. 2003] is a framework developed for the criation of multiplatform game engines. The goal of this framework is to allow the construction of game engines, both to simpler or complex games, executing in personal computers, handhelds and mobile devices. It is expected that the componentry provided by Amphibian allows the reuse of logical objects from a game, besides the structural objects like, for example, viewers, client-server architecture and persistence mechanisms. The idea is to allow the developer to create the logics of a game and that this logic should be reused to create a version to personal computers, handhelds and/or mobile phones. Each platform should have to be specifically implemented, but the game logics are kept. In this way, to create a game engine, it's only necessary to choose the main components, the game logics and create the multimedia resources. There is a note about this framework: unfortunately Amphibian does not allow a massive amount of players connected to the games created with it.

A commercial game with interperception-related characteristics which appeared recently is Super Paper Mario [Nintendo 2007]. A game where players can exchange between 2D and 3D graphics at run time, besides the capability of rotate the 2D perspective - which by itself creates a three-dimensional notion. This game is not a trully interperceptive game because it is not a multiuser game, so, there is no need to translate messages at the same time to different interfaces, although there should be a demand to the game engine to handle with both two and three-dimensional messages.

Interperception is a recent concept, which is related to translation of messages to different interfaces of a same environment, to multiuser (even massive) applications. For the fact of being recent, there are yet few games which implement this concept, but by the related works presented here, we can see that there is a progressive convergence to it.

3 System Architecture

An abstraction of a shared virtual environment can be accessed in many, different graphical interfaces (like 2D, 3D or textual ones) by using the inter-perception paradigm, which provides to the users from any of these interfaces the sense of being in the same environment of each other. This is achieved by way of treating two important aspects:

- A unified server that can resolve messages from different kind of environments;
- A unified model for each environment whose interfaces will be "merged";

The unified server receives messages from all clients connected to it and abstracts the original environment. Through this server, all the connected users can share text messages, audio streams and avatar [Burdea and Coifet 2003] movement. For example, if there is a user "A" connected to the server in the 2D interface and another user "B" with a 3D interface, the server would receive and handle the messages from both. To provide the right execution of each environment, the server converts the messages that will be transmitted between the interfaces, according to the proper syntax and semantic of each interface.

In order to merge these interfaces into one environment obtaining a single shared space accessed by all players who are connected by the unified server, each interface of this environment must present a similar model. In this way, if there exists an interface I_a whose model represents a green house, to create an inter-perceptive approach with an interface I_b, the interface I_b must have a model with the green house. The positions, orientations, dimensions of each element that composes the model of each interface must be respected. By respecting these measures, one allows the green house from the previous example to be perceived in the same way by all people using the shared space provided by the inter-perception. In other worlds, the inter-perception creates a shared space where people can perceive the same environment, although using different interfaces.

The main functionalities of inter-perception applied to games are shown in the architecture presented in Figure 2. This architecture is based on the client-server model [Tanenbaum and van Steen 2001] and uses the framework H-N2N [Burlamaqui et al. 2006]. This framework allows the creation of large scale environments [Greenhalgh], which supports large amount of users. The components *SlaveServer*, *GroupServer* and *AplicationServer* shown in figure 2 come from the H-N2N. The *ApplicationServer* component awaits client connections. The *GroupServer* manages the message exchange amongst clients. Finally, the *SlaveServer* component is responsible for the communication with the client (Game Client A, B, ..., N).

The Portal component acts on the conversion of the messages. When a message coming from a client arrives on the *SlaveServer*, it is sent to the *GroupServer*. The *GroupServer* sends this message to all *SlaveServers* attached to it. When the message arrives on a *SlaveServer* which is attached to a destination client, the message goes to *PORTAL* to be unpacked and analyzed. *PORTAL* finds the type of message and then applies the necessary transformation on it. Thus, it is ready to be sent to the destination client. For each different environment, there exists a Portal.

The interfaces(Game Client A, B, ..., N) shown in the Interperception architecture represents any kind of client application. They are originally developed to be used in virtual environments, but in this article we substitute the virtual environments by games. Each *Game Client* presented in this architecture represents a different interface, though they are different abstractions of the same environment model provided with inter-perception.

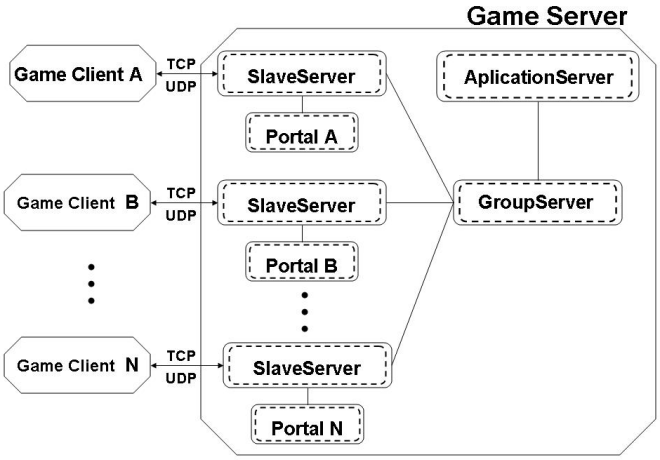


Figure 2: Interperceptive Game Architecture

3.1 Game architecture for interperceptive games

The game-client used to validate our approach in this work follows the architecture shown in Figure 3. The heart of the game is the *Controller* module. It receives requisitions of the other modules and executes them. The *Input/Output* (I/O) *Manager* sends information

to the *Controller* about all devices integrated with the game, that includes joystick, keyboard, mouse and other devices. The *Controller* handle these inputs and may generates an output that will be execute by the *I/O Manager*, like a sound that would be played. The *Network* module acts as a bridge between the game and the server. The Network module brings messages about the new positions of all player avatars and the player communications among each other. This module also carries the same messages to be sent to the other game-clients.

The *View* module acts on the render of the game scenes. This module shows the game to the player. The *Data* module contains all information needed to the *View* works. The data stored in the *Data* module are the type of avatar chosen by each player, the previous position of his avatar on the game world and other relevant information.

We have named some components with an *N* in Figure 3 and Figure 4 to show some components that need to change depending on the game. In Figure 3, we have the *View N* and *[*hData N components. It is obvious that the *View* module needs to be changed depending on the application, and the *Data* module needs to change in order to support the view's change and the *Controller* change as well. In Figure 4, we specify the *Controller* module showing that it is strongly linked with the application. As can be seen in Figure 4, only the *Game Rules* module does not need to change. As an example, let us take the *Table Soccer* InterP, better described in section 4. In a 2D representation, the *View* needs only do render 2D graphics, no depth buffer was needed nor texture buffer. But, in the 3D representation, there are 3D graphi cs with textures, lights and depth that need to be handled by the view.

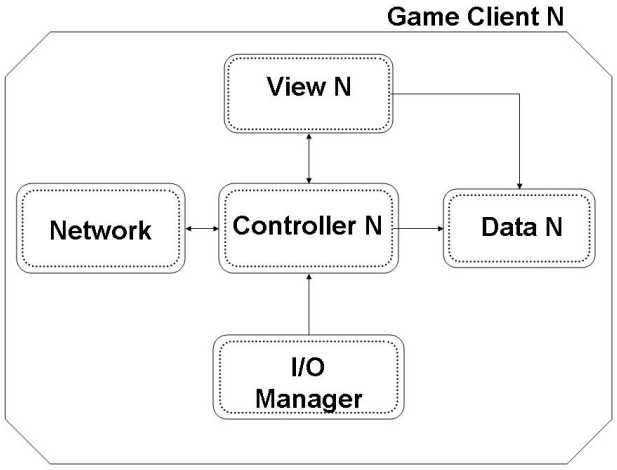


Figure 3: Game-client architecture

As the *Controller* is the more important module in this architecture, we explain it in more details here. The architecture of this module is shown in figure 4. The *Game Rules* module contains the descriptions about the rules that govern the game, for example how many points a player must do to win the game. The Event Manager receives and handles the events. Events may come from the *I/O Manager* module (a player push a button on the keyboard and trigger an event in the game), from the Network (an event generated by a player affects the others) or from the *Collision Manager* module (a player collides with an object and trigger an event).

The *Kernel* module works managing all the components of the *Controller* module. It compiles the results of all internal and external game events based on the *Game Rules* module, it makes the communication between the Controller and the network interface, and it makes the communication between the *Controller* and the *Data* module, and, finally, it commands the main state machine of the game.

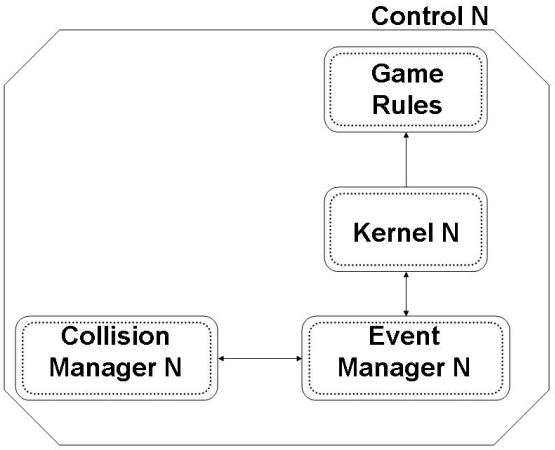


Figure 4: Game control architecture

4 The Interperceptive Table Soccer

The InterP(an acronym of inter-perception) architecture shown in the previous sections is used in the development of a game that uses two interfaces (2D and 3D), in order to validate the InterP architecture as a game. Here we show the communication between both interfaces and some characteristics of games that use this model. We developed a game that simulates a *Table Soccer* game, that we also called *Button Soccer* InterP or Futebol de Botão InterP. It consists in a environment that provides to the users a way for choosing which interface they want to play. In the 3D interface, where they will have good graphics, lights, textures, 3D sound. Or in the 2D interface that requires less memory, but has an isometric view with 2D graphics. Besides, the chosen of what interface is better to the user will not make impossible a game between a user in the 2D interface and another in the 3D one.

We used the *Model-View-Controller* (MVC) [Gamma et al. 2000] designer pattern in the development, with the Java language and the Java3D API [community Java3D 2007] in the 3D interface. For the 2D interface, we used part of the engine proposed in the book *Developing Games in Java* [Brackeen 2003].

4.1 Logical structure of the Interperceptive Button Soccer

The logical structure of the game is shown in the Figure 5. In this figure, we did not specify the network communication because it is a separate block and does not make any difference to the main logical loop of the game.

In our game, we have the traditional mode (player mode), where the players dispute a *Button Soccer* game and an additional mode, the observer mode. In the observer mode, the people can only observe the game, chatting with the players or with other observers, providing a better interaction between people who play. The logical structure of the game in the player mode is shown in Figure 6 and for the observer mode is shown in Figure 7.

In the second state, the application establishes the connection with the InterP server. Then, it waits for initialization messages. These messages refer to information about the map, sprites, and the mode (observer or player) of this user. If there are two players already (players in 2D or 3D), the server responds with a message to this user. That indicates the user to enter in the observer mode. Whatever, this user can enter the game in the player mode.

4.2 The Player Mode

In the player mode, the application waits for player inputs, computes possible collision cases, and sends the movement information to the InterP server to broadcast the information to all the other

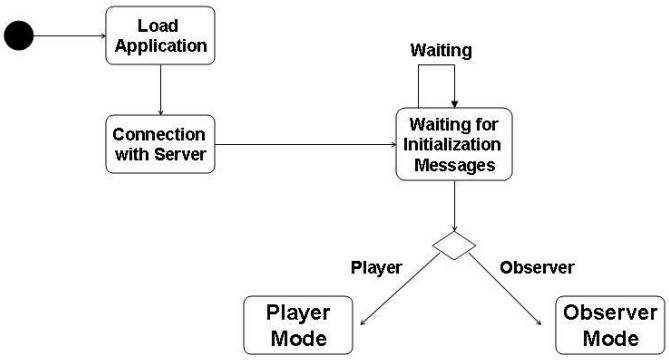


Figure 5: The logical diagram of InterP Button Soccer

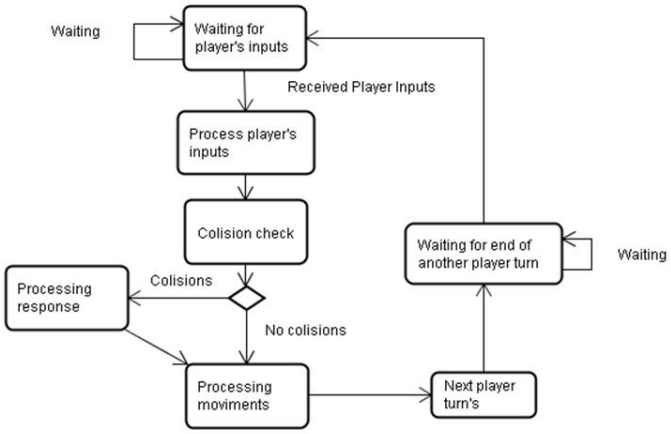


Figure 6: The logical diagram of InterP Button Soccer in the Player Mode

users. An important fact is that collision tests and processing are made in the client, so only the movement information is sent to the InterP server. At the end of each turn, the client sends a message to the server indicating that the client has finished his turn so the other player can make his move. When the other player is moving, the current player only watches to the game or sends text messages to other users.

4.3 The Observer Mode

In the observer mode, the user has an avatar in the environment that allows him to interact with the players or with another observers in the same game. This kind of user can not enter in the game field, they stay only in special locations in the game environment in order not to disrupt the player's concentration. The observers can move the camera to watch the game and can chat with the players or with other observers in 3D or 2D environment. The application yet computes the collision cases, sending only movement's messages to the server.

4.4 The 2D Interface

Figures 8 and 9 show the two modes of the 2D interface developed based in the engine proposed in the book *Developing Games in Java*, that provide us some easy structures that help us to modularize the development of the game. The main concepts that we use in this game are:

- Easy way to render the 2D tile maps;
- Some hierarchy based classes like sprite, animation and creature, that provides animation too;
- A modularized class that handles the mouse and keyboard events;

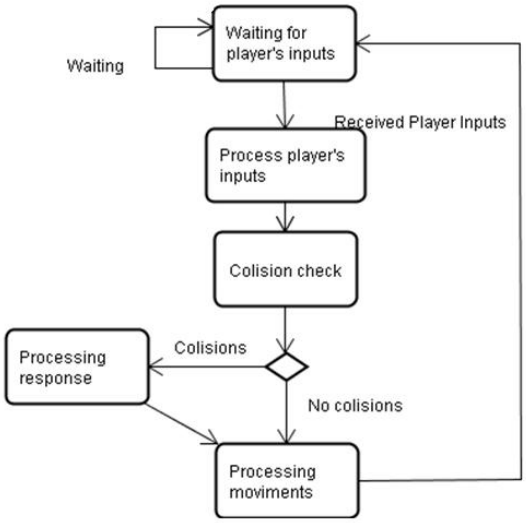


Figure 7: The logical diagram of InterP Button Soccer in the Observer Mode

- A basic collision handler;

By starting with that engine, we noticed that some characteristics need to be changed, because we had to use an isometric engine in order to make possible the correspondence between the 2D and 3D interfaces. Because of the isometric environment, we need to modify the collision handler in the render and also in the tile map. We add some features in the tile map like background music, special tile proprieties like the possibility of rendering the tiles behind or in front of the player and not-collidable tiles in order to allow the depth sensation.

Figure 8: The interface of InterP Button Soccer on the 2D observer mode

The collision engine was changed to be able to compute the right collision response by using the classic physics relations of movement transference and collisions between circles and another 2D corpses, as we see in Figure 10. The left image shows how the collision module works in a collision between two circles, the "V" vectors are the velocities, the V_b and V_c vectors are the projection of the velocities in the collision axis, the V_a and V_d vectors are the projection of the vectors in the axis perpendicular to the collision one. The right image shows the result of the collision calculus.

We decided to calculate the collisions at the client side to produce transparency between the interfaces. So a collision in 2D is checked

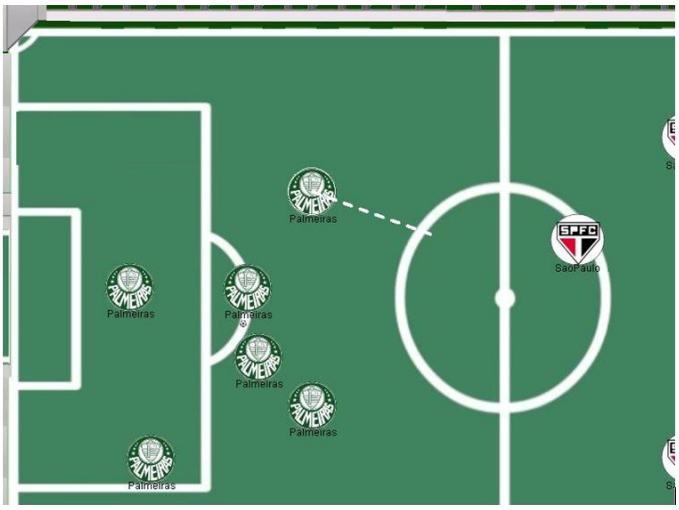


Figure 9: The interface of InterP Button Soccer on the 2D player mode

in the 2D local environment, so the client calculates the collision effects and sends to the server the next position of the sprite that is moving until the movement is not finished. This generates only a single traffic in the network. Some effects like gravity and rotation are ignored because we did not use jumps or rotations in this game.

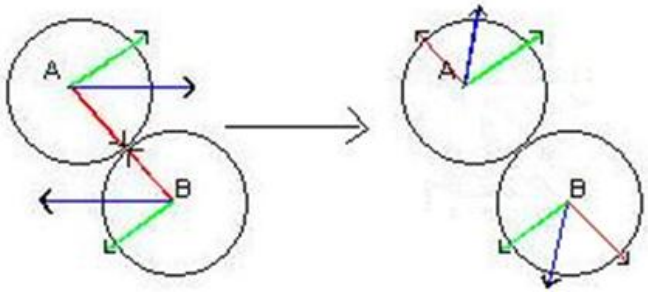


Figure 10: Collision Schema

The tile collision was used to insert some feats of the game in the tile map. We have developed a tool to edit and save 2D tile maps in our own format, in order to make it easier the map development, loading and debugging. We named this tool as *Tile Map Builder*. As we see in the figure 11, the tool provide a friendly environment to develop 2D maps in tree layers. The tool yet allows the changing of the map's special features and saves the map in our own binary format. In the construction and loading of the tile map we also used a XML configuration file to specify the main features of the map. Like where is the map file, the map's background music, the initial position of the buttons, and the spawn position of the observers. This XML is read every time a user loads the map and help us to make easier to change most of the map's proprieties.

4.5 The 3D interface

The 3D interface modes are showed in the figure 12, that shows the player mode, and figure 13, that shows the observer mode. In the 3D interface we do not use any know engine. In this case, we have used only the Java 3D API to try to develop a small game engine to this application, with still good results. We used GMax [TurboSquid 2007] to edit and make our models, in order to export then in the .OBJ format and then to convert to the MSH Format [Josivan Xavier 2006] that provides a better compression than the .OBJ format.

3D collision is made with the same method of the 2D Interface. To provide a better integration between the 2D and 3D, the map, buttons and observers' avatars are built trying to keep the same shape

Figure 11: The Tile Map Builder tool interface

Figure 12: The 3D interface of Futebol de Botão InterP on the player mode.

that is shown in 2D and with a special proportion in order to make it easier to convert the complex 3D movement into a 2D one.

Besides the 3D Interface allows the users to do some different interactive actions, that is not possible in the 2D interface. Like a first person view and a third person view of the environment, when inside the 2D interface, we have only the third person view. We have done this to show that some actions in 3D are not feasible in 2D. But we do not need to handle then in the InterP architecture because the existence of this action does not make any difference in the game play. In some cases we need to create other actions in the 2D interface in order to make the game balanced between players in the different interfaces.

3D scenes are built at the same time of the 2D maps and both interfaces use the configuration XML file to make it easier to configure the environment features like the place where the .MSH files that composes the environment is, textures, the physics information of the buttons like mass, initial position and collision bounds.

4.6 Convert positions between the interfaces

To convert a position from a dimension to another we define some rules:

- In the 2D environment, each tile from the map is square shaped and has pre-defined dimensions. In the *PercepCom2D* we define these dimensions as 64×64 pixels;
- The position of an avatar in the 2D interface is defined relative to the top left side of the image (the origin);

Figure 13: The 3D interface of Futebol de Botão InterP on the observer mode.

- In the 3D environment, position is defined according relative to the center of the avatar (the origin);
- The environments must have the origin in the same position;
- The area of the environments must be proportional to all objects.

With these rules, we define a set of equations to convert a point from 2D to 3D and the opposite. A position in the 3D environment is defined as a tuple ($3d.x$, $3d.y$, $3d.z$) and a point in the 2D is defined as a tuple ($2d.x$, $2d.y$). The equations for implementing these transformations are:

$$
\begin{aligned}
3d.x &= \frac{2d.x + \frac{T_S}{2}}{R} \\
3d.y &= predefined \\
3d.z &= \frac{2d.z + \frac{T_S}{2}}{R}
\end{aligned} \tag{1}
$$

where T_S is the length of the Tile Side and R is the ratio between the length of the 2D environment area and the 3D environment area and the *predefined* value depends on the avatar size;

With Equations 1, if an avatar in the 2D environment is at position (64, 64), in the 3D space it will be in the position (1,92; pre-defined height; 1,92). Coordinate X in the 2D interface is mapping to X in the 3D one. Coordinate Y in the 2D is mapping to Z in the 3D. The Y (height) of 3D is pre-defined because motion in vertical direction is not implemented in the *PercepCom2D*. Half of T_S defines a point in the center of the sprite of the 2D avatar and this measure is needed to define the center of a 3D avatar. The ratio R defines the rate of pixels, that is the measure unit in the 2D environment, to meters, the measure unit in the 3D environment. In the *PercepCom* system, we define this ratio to be 1 pixel for each 2 cm.

5 Results and Tests

This section shows results of a test that was done with two users, one of them with the Brazilian team *São Paulo*, using the 3D interface, and the other with the team *Palmeiras*, using the 2D interface. This experiment spent 6 minutes, divided in two rounds of 3 minutes each. At the end of the game we evaluated three aspects that we considered important to our experiment. The aspects are:

- If the communication between the two interfaces is made without problems.
- If the players did not notice any difference while playing with another player in a different interface.
- If the game is played with no advantages provided by the interface.

The first aspect is evaluated. Communication between the interfaces is made with no problems. This is possible because in the InterP architecture the client application does not need to care about performing conversions between the interfaces. The client only sends and receives movement messages, it does not get unsynchronized. Then, the scene viewed in both interfaces is synchronized too.

In the second test, the transparency of our system was validated. During this game match, the players showed no care about the other player interface. Both of then are playing in different interfaces, the game play and the interactions like chatting is the same like players of same interfaces. That was possible because of the interface tools developed in each interface to make the game equal to the users.

On the third test, we noticed an advantage in the 3D interface that consists in the fact of the 3D client interface has a bigger vision of the game, while the 2D one needs to move the camera to have the same vision. Another possible problem that we noticed is related to collision detection. In this game, we only have simple collision tests that do not use rotations, collisions in height or another complex collision computation that can be done in 3D. These tests can make both systems (the 2D and 3D interfaces) much more complex in order to show the same results in both interfaces.

The game interfaces for the results can be seen in figure 14. This figure shows a game diputed on the player mode. The top image shows the 2D interface that was used by the player with the Palmeiras team. The bottom image shows the 3D interface was used by the player with the São Paulo team. As can be seen on this figure, in the momement captured for creates this images, the two players see all the buttons in the same position and orientation. Same the player on 2D interface (Palmeiras team) can not see the buttons of the player of 3D (São Paulo) he kwons that they are in the other side of the camp.

Finally, we made tests with the players on the observer mode. The results can be seen on the figure 15. In this this figure a player from the 2D interface(the top image) are talk with the other players of the observer mode. This player send a message that appear above his head. This message also appear on the 3D interface of the observer mode (the bottom image) allow the players from this interface see the message too.

6 Conclusions

The main contribution of this work is the model definition of interperception for games. This new approach brings to the games the concepts of accessibility and shared interface. It also presents a game server built following a framework that provides a massive multi-user game environment.

Another contribution that we noticed in this work is an approach to the problem of communication between the same applications made with two different game engines. In this work, we have done the communication between a 2D application and a 3D application in two different engines. Based on the results, we can try to generalize the concept of inter-perception to enclose the concept of communication between same applications made in any game engine as well.

Finally we noticed that in the Section 3.1 where we show the architecture that we used to inter-perceptive games in the Figure 4, the *Kernel* need to be a specific module, depending on the application. However the main objective of a kernel is to be invariant depending on the applications. We have had some problems while trying to construct an invariant kernel because there are many features like event handling that need to be changed with the application.

References

AZEVEDO, S., BURLAMAQUI, A., DANTAS, R., SCHNEIDER, C., GOMES, R., MELO, J., XAVIER, J., AND GONÇALVES, L. M. G. 2006. Interperception on shared virtual environments.

BARTLE, R., 2007. Early mud history. `http://www.ludd.luth.se/mud/aber/mud-history.html`, July.

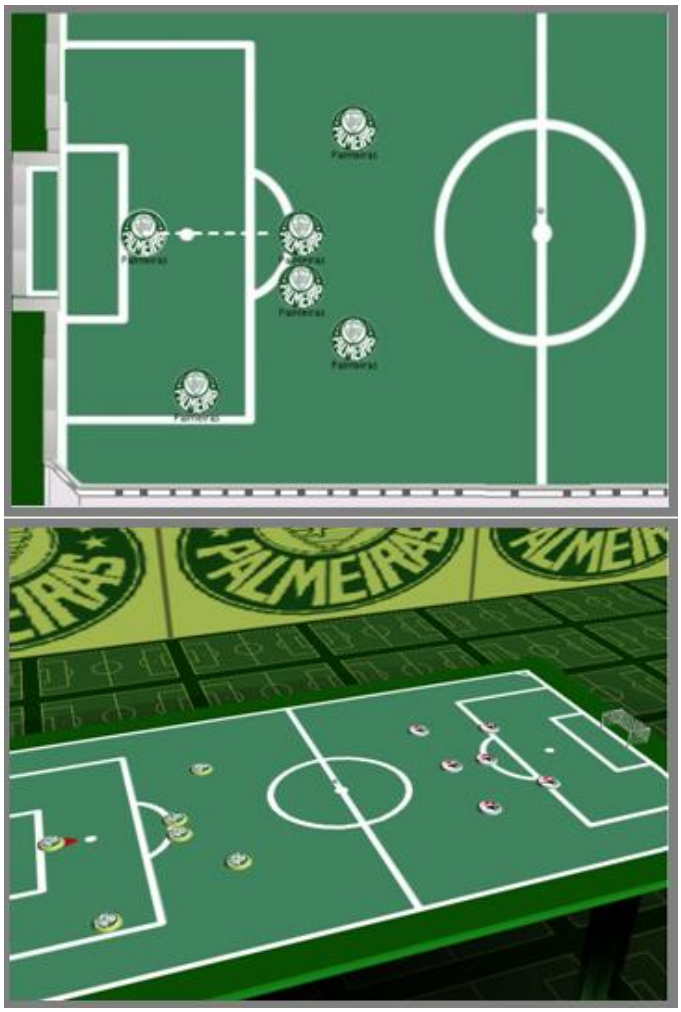


Figure 14: Test results on the player mode. Top image shows the game in 2D interface. Bottom image the 3D interface

Figure 15: Test on the observer mode. Top image shows the 2D interface. Bottom image shows the 3D

BITTENCOURT, J. R., GIRAFFA, L. M., AND SANTOS, R. C. 2003. Criando jogos computadorizados multiplataforma com amphibian. In *Proceedings of II Congresso Internacional de Tecnologia e Inovação em Jogos para Computadores*.

BRACKEEN, D. 2003. *Developing Games in Java*. New Riders Games.

BURDEA, G., AND COIFET, P. 2003. *Virtual Reality Techinologies, Second Edition*. IEEE Press.

BURLAMAQUI, A. M., OLIVEIRA, M. A. M. S., GONÇALVES, L. M. G., LEMOS, G., AND DE OLIVEIRA, J. C. 2006. A scalable hierarchical architecture for large scale multi-user virtual environments. In *Proceedings of IEEE International Conference on Virtual Environments, Human-Computer Interfaces, and Measurement Systems*.

COMMUNITY JAVA3D, 2007. Java3d api. `https://java3d.dev.java.net`, June.

GAMMA, E., JOHNSON, R., AND VLISSIDES, J. 2000. *Padrões de projeto*. Bookman Companhia.

GREENHALGH, C. Large scale collaborative virtual environments. *Springer*.

HAN, J., IN, H. P., AND WOO, J.-S. 2005. Towards situation-aware cross-platform ubi-game development. In *Proceedings of Pervasive2005*, 8–13.

JOSIVAN XAVIER, AQUILES BURLAMAQUI, L. G. 2006. Diminuindo o tempo no armazenamento e recuperação de dados geométricos em aplicações web em tempo real no java3d. In *Proceedings of Workshop de Iniciação Científica do XIX SIBGRAPI*.

LINDT, I., OHLENBURG, J., PANKOKE-BABATZ, U., GHELLAL, S., OPPERMANN, L., AND ADAMS, M. 2005. Designing cross media games. In *Proceedings of the International Workshop on Gaming Applications in Pervasive Computing Environments - PERGAMES*, 62–66.

NINTENDO, 2007. Super paper mario. `http://wii.nintendo.com/site/spm/`, June.

TANENBAUM, A. S., AND VAN STEEN, M. 2001. *Distributed Systems: Principles and Paradigms*. Prentice-Hall.

TURBOSQUID, 2007. Gmax. `https://java3d.dev.java.net`, June.

WEST, T. 2007. *RuneScape: The Official Handbook*. Scholastic Library Publishing.

Análise da plataforma J2ME através de um estudo de caso na área de jogos multiplayer

Fernando Bevilacqua
Andrea Schwertner Charão
Cesar Pozzer
Programa de Pós-graduação em Informática - Universidade Federal de Santa Maria

Resumo

Applications for mobile devices have increased their complexity level over the years, resulting in more sophisticated software, like multiplayer games. Within this context, this paper intends to raise useful information about J2ME development experience, through a case study on implementing a multiplayer game. The case study shows advantages, disadvantages and characteristics of J2ME, focusing on multiplayer game development. The paper also shows a comparison of advantages and disadvantages between J2ME and other two development platforms, BREW and Flash Lite, intending to improve the J2ME analysis. During the game implementation, optimization techniques are used to reduce the game memory usage; those techniques are focused in image optimization and code obfuscation.

Keywords:: Multiplayer, Celular, J2ME, Game, Flash Lite, BREW

Author's Contact:

{fernando,andrea,pozzer}@inf.ufsm.br

1 Introdução

Os jogos para telefones celulares estão evoluindo em um ritmo acelerado nos últimos anos. Como inicialmente os aparelhos possuíam poucos recursos, os primeiros jogos eram rudimentares e pouco numerosos. Com a crescente incorporação de características aos aparelhos, os jogos para celular foram se tornando mais complexos, com gráficos e sons mais elaborados e uma interatividade maior do que seus antecessores, possibilitando inclusive a interação entre múltiplos jogadores. A demanda e a oferta de jogos também aumentaram, de modo que atualmente o mercado de jogos para celulares constitui uma área rentável e em plena expansão [Wisniewski et al. 2005].

Mesmo com os recentes avanços em jogos para celulares, percebe-se que os jogos multi-jogador ainda não são amplamente explorados nestes dispositivos móveis. Dentre os motivos que podem explicar esse fato, estão a deficiência de meios sofisticados para conexões entre aparelhos (apenas os protocolos mais básicos de comunicação estão disponíveis na maioria dos dispositivos) e o estado ainda pouco consolidado das plataformas de desenvolvimento deste tipo de jogo. Neste sentido, as experiências de análise e utilização das ferramentas existentes constituem subsídios importantes para o avanço e a disseminação de conhecimentos nesta área.

Considerando o cenário acima delineado, este artigo expõe a experiência de desenvolvimento de um jogo de estratégia multi-jogador para telefones celulares, utilizando uma das principais ferramentas de desenvolvimento para dispositivos móveis: a plataforma J2ME (Java 2 Micro Edition) [SUN MICROSYSTEMS 2006]. O jogo escolhido para este estudo de caso é o Stratego [Wikipedia 2006], um jogo de estratégia popular e, até onde foi possível constatar, ainda inexistente para telefones móveis. Com esse artigo, pretende-se reunir informações relevantes sobre o desenvolvimento desse tipo de aplicação nessa plataforma, a fim de contribuir com outros desenvolvedores que atuam nesta área.

2 Trabalhos relacionados

Os jogos multi-jogador para telefones celulares, bem como a análise entre plataformas de desenvolvimento, têm sido temas de trabalhos e publicações recentes. Alguns desses trabalhos foram analisados e são brevemente comentados a seguir.

O primeiro trabalho analisado foi o jogo BlueGame [Rabelo et al. 2005]. Trata-se de um jogo de combate que pode ser jogado por duas pessoas. O jogo foi implementado na linguagem C#, com a plataforma de desenvolvimento .NET Compact Framework. Este jogo utiliza um modelo *peer-to-peer* para comunicação, com transmissão de dados serial através da tecnologia Bluetooth. Não está dentro do escopo do presente trabalho realizar análises sobre a implementação de jogos utilizando-se a linguagem C#, porém uma análise sobre utilização de Bluetooth é citada na seção 3.3.2.

O segundo trabalho analisado foi o jogo *The Right Way* [Hedlung and Maxe 2004]. Este é um jogo de estratégia que pode ser jogado por várias pessoas, na qual cada uma controla um personagem, que está posicionado em um labirinto. O jogo utiliza um modelo cliente-servidor para comunicação, através do protocolo HTTP 1.1. O cliente e o servidor foram implementados utilizando-se a linguagem Java. Sobre os recursos utilizados para o desenvolvimento de *The Right Way*, existe um estudo realizado por [KARLSSON et al. 2003], que mostra uma comparação entre as plataformas J2ME, Symbiam e BREW, fundamentada nas experiências de desenvolvimento do grupo de pesquisas CIn/CESAR, em relação ao uso dessas plataformas para o desenvolvimento de jogos. No trabalho, são mostradas comparações entre as diversas características de cada uma das plataforma, como gerenciamento de I/O, processamento gráfico, processamento de som e conectividade. A análise possui um cunho técnico e é voltada, conforme informações do próprio texto, aos desenvolvedores de jogos para sistemas móveis, informando sobre peculiaridades encontradas em cada uma das plataformas, como disponibilidade de execução do pressionamento de mais de uma tecla do dispositivo simultaneamente, por exemplo.

O terceiro trabalho analisado é uma proposta de arquitetura que dá suporte ao cenário de pequenos jogos para dispositivos móveis [Pereira 2006], utilizando Bluetooth para comunicação. Conforme explanado pelo autor, os jogos multi-jogador para celulares são uma evolução dos jogos dessa categoria para PC, o que é comprovado pelo lançamento de iniciativas comerciais nesse sentido, como o aparelho Nokia NGage [NOKIA CORPORATION 2006]. Através de um paralelo com jogos MMOG (*Massively Multiplayer Online Game*), o autor cita que a união desses com jogos móveis cria um novo conceito de jogo, os MMMOGs (*Massively Mobile Multiplayer Online Games*). Outros estudos também foram realizados sobre jogos multi-jogador para dispositivos móveis, comprovando a existência da uma preocupação com tópicos dessa área, como no trabalho de [Sacramento et al. 2005], que propõe um *framework* para o desenvolvimento de jogos desse estilo (multi-jogador para dispositivos móveis). O objetivo principal do referido trabalho é proporcionar aos desenvolvedores de jogos um acesso transparente entre os aparelhos, a fim de que não haja preocupação em questões específicas no processo de comunicação, como a forma com que as informações são trocas entre os dispositivos. O *framework* é desenvolvido em Java e provê um conjunto de funcionalidades sobre o protocolo HTTP, como envio de mensagens, validação por login e senha, validação de usuário por grupo, envio de requisições, obtenção de uma lista de informações pertinentes (pontos, posição no *ranking*, apelido, etc).

3 Plataformas de desenvolvimento

As seções que seguem apresentam, respectivamente, uma breve descrição da plataforma BREW, que é baseada em C/C++, uma descrição da plataforma Flash Lite, que é uma das plataformas mais recentes, e uma descrição mais detalhada da plataforma J2ME. Em seguida, explana-se sobre o jogo escolhido como caso de estudo da plataforma J2ME.

3.1 BREW

A plataforma BREW (*Binary Runtime Environment for Wireless*) [QUALCOMM INCORPORATED 2006] foi criada e lançada em 2001 pela empresa Qualcomm. Esta plataforma destina-se ao desenvolvimento de *software* para aparelhos celulares com tecnologia CDMA (*Code Division Multiple Access*). Nessa plataforma, os desenvolvedores podem escrever aplicações na linguagem nativa (C/C++). Estas aplicações serão compiladas e executadas em um *chip* à parte do processador principal do aparelho, o que garante um maior desempenho de execução. Além de C/C++, o programador também pode utilizar outra linguagem cujo ambiente de execução seja suportado, como Java e Flash [ADOBE SYSTEMS INCORPORATED 2006b].

Antes do lançamento dessa plataforma, os desenvolvedores de aplicações para celular necessitavam programar características específicas para cada aparelho [Barros 2005]. Com o lançamento da BREW, a Qualcomm uniformizou o ambiente de execução das aplicações em diferentes dispositivos. Essa unificação se deu também no modelo de negócios adotado pela Qualcomm, que passou a oferecer recursos para divulgação e venda dos jogos. Nesse modelo, os jogos criados por desenvolvedores são primeiramente enviados à Qualcomm, que realiza testes sobre o jogo. Depois que a aplicação está testada e possui garantia de funcionamento e qualidade comprovada, ela é enviada para comercialização na rede de distribuição, composta pelas operadoras de telefonia celular filiadas à Qualcomm.

Os desenvolvedores de jogos que utilizam a BREW ficam obrigatoriamente vinculados ao modelo de negócios da Qualcomm e das operadoras da rede de distribuição. Embora esse modelo de negócios diminua a porcentagem de pequenos desenvolvedores investindo nessa plataforma, há algumas vantagens tanto para o usuário final quanto para a empresa que veicula seus jogos. A principal delas é que o *download* de todos os jogos é efetuado através da rede de distribuição, que é um ambiente comprovado e seguro, no qual o usuário pode confiar na hora de realizar suas compras. Esse meio permite, também, um controle maior sobre o que é instalado nos aparelhos celulares, o que previne a pirataria de jogos.

3.2 Flash Lite

A plataforma Flash Lite [ADOBE SYSTEMS INCORPORATED 2006a] é uma parte da plataforma Flash, da empresa Adobe Systems Incorporated, voltada a dispositivos móveis. Ao contrário do Flash Player, responsável pela execução de conteúdo Flash em microcomputadores [ADOBE SYSTEMS INCORPORATED 2006b], a plataforma Flash Lite oferece um ambiente mais otimizado em relação à utilização de recursos, como memória e processamento. Firmada sobre as capacidades do Flash Player 7, a plataforma Flash Lite, em sua versão 2.1, apresenta recursos como comunicação por *sockets*, manipulação de XML, utilização de gráficos vetoriais e recursos multimídia (como visualização de imagens, áudio e vídeo), armazenamento persistente de dados e suporte a Unicode [ADOBE SYSTEMS INCORPORATED 2007].

Conforme análise divulgada por [Linsalata and Slawsby 2005], a barreira criada entre desenvolvedores inexperientes ou com pouco conhecimento em desenvolvimento de aplicações para dispositivos móveis pode ser contornada com o uso do Flash Lite [Linsalata and Slawsby 2005]. Fazendo uso do ambiente integrado para desenvolvimento de conteúdo Flash, desenvolvedores que já possuem conhecimento e habilidades na plataforma para PC podem, também, utilizar esse conhecimento para desenvolverem conteúdo para dispositivos móveis. Essa redução da quantidade de conhecimento técnico necessário para o desenvolvimento de aplicações para aparelhos móveis aumenta o número de potenciais desenvolvedores que possam utilizar a plataforma, como é o caso de *designers* com conhecimento na plataforma Flash [Linsalata and Slawsby 2005].

3.3 J2ME

A plataforma J2ME [SUN MICROSYSTEMS 2006] é um conjunto de tecnologias e especificações voltadas para o desenvolvimento de aplicações para sistemas embarcados com recursos limitados, como celulares e PDAs (*Personal Digital Assistants*) . Esta plataforma compreende interfaces de usuário, modelos de segurança, protocolos de comunicação em rede e outras funcionalidades que, quando combinadas, constituem um ambiente de execução Java otimizado para uso de memória, processamento e operações de entrada e saída.

Lançada em 1999, a J2ME é composta basicamente de duas camadas dispostas sobre a máquina virtual Java: a camada de configurações (*configurations*) e a camada de perfis (*profiles*). A primeira se refere à configuração mínima exigida por um conjunto de dispositivos que apresentem características de *hardware* semelhantes. Aparelhos fortemente distintos, como geladeiras e telefones celulares, têm suas características agrupadas e definidas em diferentes configurações. A camada de perfis provê o suporte para as necessidades específicas de dispositivos de uma determinada família, garantindo interoperabilidade das aplicações entre estes aparelhos.

3.3.1 Configurações e Perfis

Uma configuração define a plataforma Java para um grupo de aparelhos com características de *hardware* semelhantes. Estão incluídas nessa definição os recursos da linguagem Java, os recursos da JVM e, também, as APIs que são suportadas. Existem dois tipos de configurações atualmente definidas: a CLDC (*Connected Limited Device Configuration*) e a CDC (*Connected Device Configuration*). A primeira define uma parcela da plataforma Java destinada a aparelhos com restrições significativas de memória e processamento, enquanto a segunda é voltada para aparelhos com maior capacidade de processamento e maior disponibilidade de recursos. Os perfis oferecem o suporte necessário para uma família específica de dispositivos. Cada configuração pode apresentar diferentes tipos de perfis.

A CLDC apresenta várias limitações nas funcionalidades do Java em comparação à plataforma J2SE (*Java 2 Standard Edition*), como inexistência de números de ponto-flutuante, limitações para manusear e tratar erros (a aplicação não pode se recuperar de alguns erros devido a especificidades de *hardware*), nenhum suporte a JNI (*Java Native Interface*) ou a reflexão. Além dessas limitações, as APIs disponíveis contêm menos classes que as APIs do J2SE, uma vez que elas ocupam uma grande quantidade de memória que não está disponível no ambiente dos dispositivos. Apenas um grupo reduzido e essencial de classes está disponível. A CDC também possui limitações, mas como se destina a dispositivos com uma quantidade maior de recursos, tais limitações são menores. Exemplo disso é que a CDC suporta importantes funcionalidades da linguagem Java, como números de ponto flutuante, JNI, RMI (*Remote Method Invocation*) e reflexão.

Um dos perfis que a CLDC apresenta é o MIDP (*Mobile Information Device Profile*). O MIDP é o perfil destinado a dispositivos portáteis, como celulares e PDAs, apresentando algumas características como:

- suporte a um subconjunto do protocolo HTTP, implementado sobre protocolos baseados em IP (TCP/IP, por exemplo) ou não baseados em IP (WAP, por exemplo);
- mecanismos para armazenamento persistente de dados no dispositivo;
- definição de um modelo de aplicação.

3.3.2 Comunicação

A API para comunicação disponível na plataforma J2ME é provida pelo *Generic Connection Framework (GCF)* [Ortiz 2003]. O GCF surgiu para uso no CLDC 1.1 como alternativa aos pacotes `java.net` e `java.io`, ambos pertencentes à família J2SE, porque esses pacotes são grandes em demasia para serem usados em ambientes de pouca memória, como é o caso de dispositivos móveis.

O GCF é composto por uma hierarquia de classes e interfaces, as quais definem métodos de conexão e execução de I/O. No topo da hierarquia, encontra-se a interface `Connection`, que é estendida por todas as demais classes. Descendo-se na hierarquia, as classes tornam-se mais complexas, específicas para determinado tipo de conexão, porém uma fábrica de conexões garante a uniformidade de acesso. Através de uma URL, o programador pode criar um canal de comunicação sobre um dos tipos de conexões disponíveis, utilizando a fábrica de conexões, expressa pela chamada de `Connection.open(URL)`. A URL é criada no formato `scheme://endereço:parâmetros`, onde `scheme` define o tipo de conexão a ser usado.

Embora existam vários tipos de conexões disponíveis para uso, algumas das classes que as implementam não são requeridas nos dispositivos móveis como parte indispensável do MIDP ou de outro *profile*. Mesmo que a disponibilidade de comunicação por HTTP e HTTPS seja requerida no MIDP 1.0 ou MIDP 2.0, o que garante ao programador a existência desses recursos em dispositivos com tais *profiles*, a sua utilização pode não ser a melhor forma de comunicação para determinados casos.

A utilização de *sockets*, por exemplo, permite ao desenvolvedor uma comunicação de rede de baixo nível [Mahmoud 2003], uma vez que os dados transmitidos não estão vinculados a nenhum protocolo pré-definido. Sendo assim, um protocolo próprio pode ser criado, o que evita *overheads* de processamento de cabeçalhos de pedido-resposta textuais, como é o caso do HTTP, por exemplo. No caso de jogos, algumas informações podem ser perdidas durante a comunicação sem grandes prejuízos ao andamento do jogo. Para tirar proveito disso, o desenvolvedor pode basear a transmissão de dados sobre UDP, que não é orientado a conexão (não há garantia de entrega do datagrama), porém apresenta melhor desempenho em relação a TCP/IP (orientado a conexão), uma vez que não há processamento extra para estabelecimentos de conexões de verificação.

Outro tipo de conexão que não é obrigatória é o Bluetooth. O Bluetooth é uma tecnologia de custo baixo, pouco consumo de energia e com curto alcance de radiofreqüência (de 1m a 100m), criado para substituir os cabos nas conexões entre dispositivos portáteis [Klingsheim 2004]. A comunicação é feita em uma banda de 2.4 Gz, utilizando-se uma técnica de sinalização chamada Frequency Hopping Spread Spectrum (FHSS). Nessa técnica, a banda de rádio é divida em 79 sub-canais, nos quais o rádio Bluetooth alterna o uso a cada 625 milissegundos. Os saltos entre os sub-canais são usados para reduzir a interferência causada por dispositivos Bluetooth próximos uns dos outros. A Java APIs for Bluetooth Wireless Technology (JABWT) define um pacote J2ME, chamado `javax.bluetooth`, para comunicação utilizando Bluetooth. Em se tratando de jogos, dispositivos móveis podem formar uma rede *ad-hoc* através de Bluetooth para permitir que seus usuários possam participar de uma partida *multiplayer*, por exemplo. Conforme testes de Klingshein (2004), a taxa de transferência de dados entre dois dispositivos Nokia 6600, utilizando Bluetooth, foi de 10 a 11 KB/s. Ao contrário da comunicação por HTTP ou por *sockets*, a comunicação via Bluetooth não possui custos de transmissão de dados *over the air*, uma vez que a comunicação é feita diretamente entre um dispositivo e outro. Isso pode se tornar uma opção atrativa de comunicação entre jogadores que não desejam gastar recursos financeiros com a transmissão de dados entre seus dispositivos.

4 Estudo de caso: jogo Stratego

Conforme se mencionou anteriormente, o jogo escolhido para este trabalho foi o Stratego[1]. Neste jogo, cada um dos dois participantes possui um conjunto de peças ao seu dispor e o jogador consegue ver apenas a graduação de suas peças, sendo que a graduação das peças inimigas permanece oculta. Cada vez que duas peças se encontram em uma mesma posição do tabuleiro, elas são mostradas a ambos os jogadores e a peça de maior graduação ganha, fazendo com que a peça perdedora deixe o tabuleiro (as duas peças saem em caso de empate).

Visando à escalabilidade do código, a aplicação foi dividida em três camadas: camada lógica, camada de comunicação e camada gráfica. A camada lógica contém todas as classes que implementam a lógica do jogo, como o que acontece quando duas peças se encontram, o que acontece quando a bandeira do adversário é capturada, etc. A camada de comunicação contempla as classes que implementam a troca de dados entre um celular e outro. Por fim, a camada gráfica abriga a classe que realiza qualquer desenho na tela. Somente essa classe está autorizada a desenhar na tela, carregar gráficos e executar funções desse tipo. Cada uma das camadas que compõem a aplicação é explanada em detalhes nas seções seguintes.

4.1 Camada lógica

A camada lógica da aplicação tem por objetivo isolar e centralizar elementos lógicos da aplicação, vinculando-as com as demais camadas do jogo. Para centralizar as variáveis que controlam a aplicação, como sinalizadores de avisos, indicadores de pausa e afins, bem como o ciclo de vida do programa, criou-se a classe `StrategoEngine`, que é a base da aplicação. Para prover uma centralização das regras do jogo, especificamente, criou-se a classe `GameBoard`. Estas classes, e as suas respectivas classes-base, são apresentadas na figura 1.

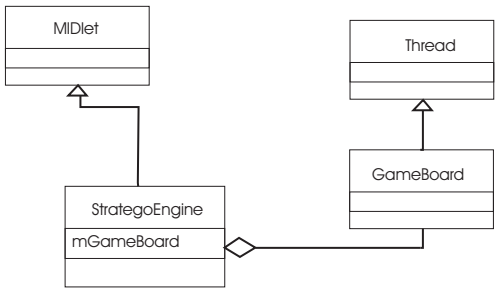


Figura 1: Diagrama de classes simplificado da camada lógica

A classe `StrategoEngine` constitui a base de uma aplicação MIDP, contendo os métodos necessários para controle de seu ciclo de vida e resposta aos avisos feitos pelo dispositivo, como interrupção do jogo em virtude de uma chamada telefônica. Estão contidos nessa classe todos os atributos importantes ao jogo e à aplicação, como estado atual da partida (pausada, jogando, dispondo peças no tabuleiro, etc.), informações sobre os dados recebidos do outro dispositivo, quem joga no turno corrente, etc. Todas essas informações são utilizadas pelas outras classes da aplicação para guiar o seu comportamento.

O outro componente da camada lógica é a classe `GameBoard`, que detém toda a lógica do jogo, desde o tabuleiro com a disposição das peças até os métodos que resolvem o combate entre elas. No momento em que é inicializada pela `StrategoEngine`, a `GameBoard` executa em uma *thread* em separado, que é o laço principal do jogo. Esse laço realiza, dentre outras funções, o controle das atividades ligadas principalmente ao tempo, como fechar as conexões de rede se o outro dispositivo está há muito tempo sem enviar quaisquer informações. Todas as entradas

[1] Frisa-se que a utilização de Stratego é apenas um caso de uso para a análise da plataforma J2ME, logo as regras do jogo não são descritas em sua totalidade aqui. É possível encontrar mais informações sobre as regras em [Wikipedia 2006]

de dados efetuadas para o dispositivo, como acionamento de uma tecla ou recebimento de uma mensagem de outro aparelho, são interpretadas pelo `GameBoard`. Essa foi uma decisão de projeto para garantir a centralização do processamento das diversas entradas que o jogo possa receber, a fim de concentrar pontos de falha críticos em poucos métodos de uma classe específica.

4.2 Camada de comunicação

A camada de comunicação desempenha uma tarefa crítica no jogo, pois uma comunicação rápida é imprescindível para que uma sessão de jogo seja satisfatória para o jogador. Visando flexibilidade, a camada de comunicação foi projetada para garantir que um jogador, de posse da aplicação, pudesse realizar todas as ações relacionadas ao jogo sem ter que realizar qualquer tipo de *download* de componentes extras ou manter em seu dispositivo várias versões do jogo. Stratego está fundamentado na idéia de cliente e servidor e, para que o jogador não necessite de *downloads* extras ou de mais de uma versão do jogo, a camada de comunicação teve de conter classes que se comuniquem de forma híbrida: hora como cliente, hora como servidor. Isso garante que o jogador não necessite manter duas versões do jogo, como uma versão "Stratego cliente"e outra versão "Stratego servidor", por exemplo.

Para a camada de comunicação criou-se, então, a classe `GameConnection`, que pode atuar como cliente ou como servidor, e a classe `Sender`, que realiza tarefas de suporte em relação ao envio de dados. Estas classes, e as suas respectivas classes-base, são apresentadas na figura 2.

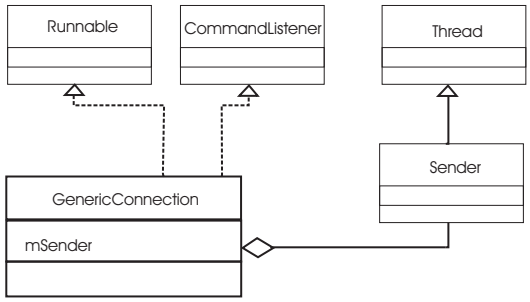


Figura 2: Diagrama de classes simplificado da camada de comunicação

A classe `GameConnection` foi projetada para possuir características de servidor e de cliente, porém sem que as demais classes da aplicação precisem utilizar uma interface diferente para realizar a comunicação. Independentemente do comportamento atual (cliente ou servidor), a classe `GameConnection` disponibiliza a mesma interface de comunicação para as demais classes do jogo.

Para simplificar a implementação, foram utilizados endereços de conexão e portas fixas nas classes de comunicação, todos criados em tempo de compilação. A conexão entre os dispositivos foi simulada no *J2ME Wireless Toolkit*, ferramenta utiliza para o desenvolvimento do jogo. Julgou-se satisfatória a análise dos resultados de comunicação obtidos com a implementação dos endereços e portas dessa forma. Ressalta-se que a implementação de Stratego constitui um estudo de caso para a análise da plataforma, logo fazer com que o jogo utilize recursos de operadoras de celular, por exemplo, para encontrar outros celulares, enfatiza elementos que estão fora o escopo desse artigo.

A alta latência de comunicação entre os dois dispositivos faz com que o jogador tenha que esperar parcelas de tempo consideráveis entre o envio e resposta de uma mensagem. Para garantir que a comunicação de dados seja tratada de forma elegante e possa ser notada pelo jogador [NOKIA CORPORATION 2003], uma tela de espera é usada. Essa tela informa ao jogador que os dados estão sendo enviados e impedem que o jogo fique sem resposta durante a transmissão de dados, o que faz com que o jogador não saiba se o sistema está enviando os dados ou se repentinamente parou de funcionar. Para conseguir enviar os dados e ainda manter a aplicação respondendo à interação do jogador, uma *thread* é utilizada. Essa *thread* é gerenciada por uma instância da classe `Sender`, criada quando `GameConnection` se inicializa. A *thread* da classe `Sender` tem como único objetivo enviar uma quantidade de bytes para o destino informado por `GameConnection`.

4.3 Camada gráfica

Para garantir que a camada gráfica do jogo apresentasse o mínimo de impacto possível em outras classes da aplicação se alguma mudança fosse aplicada às suas classes, todos os seus elementos foram projetados para possuírem pouca ou nenhuma lógica de jogo. Partindo da regra que somente as classes da camada gráfica estão autorizadas a desenhar no *display* e, também, garantindo que essas classes não contenham lógica da aplicação, as classes dessa camada podem sofrer alterações drásticas de código. Por exemplo, se algum dispositivo móvel necessitar de comportamentos e métodos especiais para que o desenho seja feito de maneira correta, grandes alterações poderão ser aplicadas às classes dessa camada, até mesmo a sua completa substituição por classes com os mesmos métodos públicos, sem que isso implique alterações nas outras classes ou lógica da aplicação. Isso é possível porque nenhuma lógica do jogo está contida nas classes da camada lógica, apenas lógica relacionada ao desenho.

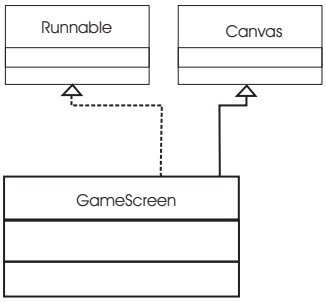


Figura 3: Diagrama de classes simplificado da camada gráfica

A classe que compõe a camada gráfica é a `GameScreen`, conforme ilustrado pela figura 3. Assim que é inicializada por `StrategoEngine`, a classe `GameScreen` obtém informações do ambiente em que se encontra, como a largura e altura máxima do *display*. Em seguida, realiza-se alguns cálculos para que o tamanho de cada quadrado do tabuleiro do jogo possa ser desenhado da maior maneira possível, a fim de que o usuário tenha uma visão clara do campo de batalha. Em seguida, `GameScreen` cria uma *thread*, que ficará ativa enquanto o jogo estiver executando. Essa *thread* tem como função realizar os desenhos na tela e, também, ouvir as entradas feitas pelo teclado. Ela é responsável por ler as entradas do teclado porque, na plataforma J2ME, a classe `Canvas` (extentida por `GameScreen`) é a responsável por manipular eventos do teclado e de ponteiros, se o dispositivo suportar [SUN MICROSYSTEMS, INC. 2006], decodificando-os. Todas as entradas que são realizadas pelo jogador são ouvidas pela `GameScreen`, que então decodifica a tecla pressionada e a traduz para um código de tecla de jogo, como `GAME_A`, `GAME_B`, `UP` e `DOWN`, por exemplo (estas constantes que designam as teclas são pré-definidas na plataforma J2ME). Cada dispositivo pode possuir o seu conjunto de teclas que representem movimentos e ações em jogos. Dessa forma, se em um dispositivo a tecla para `FIRE` for a tecla de número 5 e, em outro dispositivo, ela for a tecla de número 9, a aplicação receberá apenas o código `FIRE`, não `KEY_NUM5` ou `KEY_NUM9`, o que garante portabilidade entre os dispositivos.

Depois que `GameScreen` decodifica qual tecla foi pressionada e qual é o seu comando de jogo, ela repassa essa informação para a classe `GameBoard`, a fim de que essa trate a entrada e responda adequadamente.

A Figura 4 ilustra o uso de todas as classes em uma partida de Stratego entre dois dispositivos.

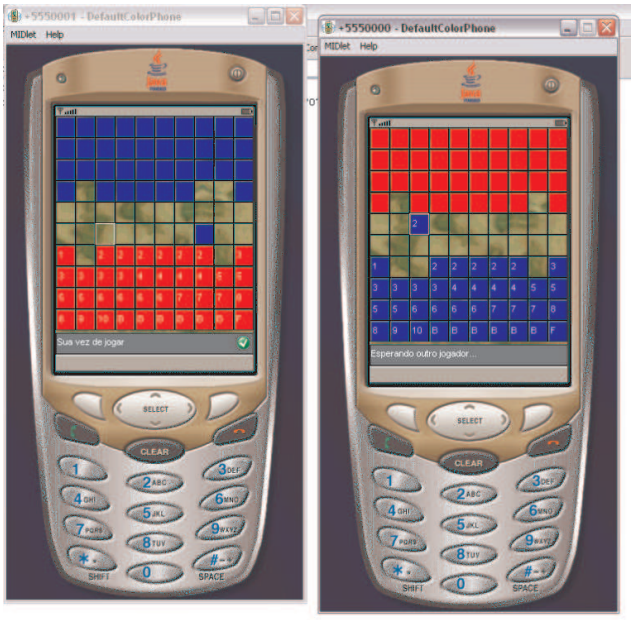


Figura 4: Sessão de jogo entre dois dispositivos

5 Avaliação da plataforma J2ME

O desenvolvimento do jogo Stratego constituiu um caso prático de utilização da plataforma J2ME para telefones celulares. Com base nesta experiência, apresenta-se uma avaliação de algumas decisões tomadas durante o processo de desenvolvimento (como formas de otimização de recursos) e, também, uma discussão sobre os pontos fortes e fracos da plataforma J2ME com base nessa experiência. Em relação à otimização de recursos, analisaram-se duas técnicas para reduzir a quantidade de recursos: a otimização de imagens e o obscurecimento de código.

Todos os testes foram simulados com a ferramenta *J2ME Wireless Toolkit*, utilizado para o desenvolvimento de Stratego.

5.1 Otimização de imagens

A otimização de imagens consiste no processo de remover do arquivo de imagem informações que sejam dispensáveis à aplicação, ou seja, elementos que se forem removidos não causarão alterações drásticas na visualização da imagem. O tipo de imagem escolhido para ser utilizado em Stratego foi o PNG, pelo fato de que esse formato é obrigatoriamente implementado pelos dispositivos que suportam aplicações da plataforma J2ME [SUN MICROSYSTEMS 2006]. Embora imagens no formato JPG ou GIF, por exemplo, sejam menores, não há garantia de que todos os dispositivos móveis tenham suporte a esses formatos.

Para otimizar as imagens de Stratego, foram utilizados dois *softwares*: o PNGout [ARDFRY IMAGING, LLC 2007] e o OptiPNG [Truta 2007]. A Tabela 1 ilustra o tamanho do arquivo resultante do empacotamento do jogo e, também, a utilização de memória durante a execução, antes e depois da otimização das imagens.

Tabela 1: Informações do jogo sobre otimização das imagens (resultados em bytes)

Otimização imagens	Tam. do JAR	Mem. max. cliente	Mem. max. servidor
Não	372.308	700.104	699.476
Sim	92.434	191.972	192.860

Conforme pode ser observado na Tabela 1, o impacto da otimização das imagens é considerável. O tamanho do arquivo resultante do empacotamento de todos os elementos que compõem a aplicação (classes e imagens) sofreu um decréscimo de aproximadamente 75%. Quanto menor o arquivo JAR resultante, menos dados o usuário terá de transferir para o seu dispositivo via *download* para conseguir obter o jogo.

Em relação à utilização de memória durante a execução do jogo, as colunas "Mem. max. cliente"e "Mem. max. servidor" indicam o pico máximo de utilização de memória da aplicação agindo como cliente e como servidor, respectivamente. Antes da otimização das imagens, o pico máximo registrado de consumo de memória foi de aproximadamente 683Kb. Após a otimização das imagens, tanto por parte do jogo atuando como cliente ou como servidor, a redução foi de aproximadamente 72%, o que resultou num pico máximo em torno de 184Kb de memória usada. Esses dados de utilização de memória durante a execução foram obtidos com o *Memory Monitor*, ferramenta integrada ao *J2ME Wireless Toolkit*, utilizado para o desenvolvimento de Stratego.

Embora Stratego faça uso de poucas imagens, a redução do tamanho do arquivo JAR resultante e também a redução da utilização de memória em virtude da otimização de imagens ficou em torno de 73%. Aplicações com uma grande quantidade de imagens PNG podem vir a reduzir drasticamente o seu consumo de memória através do uso de ferramentas de otimização.

5.2 Obscurecimento de código

O obscurecimento de código consiste em executar sobre o código-fonte da aplicação alguma ferramenta cuja função é obscurecer o conteúdo analisado, tornando-o pouco compreensível ou ilegível para humanos. Embora essa técnica seja utilizada para proteção dos arquivos compilados, a fim de evitar ou dificultar a descompilação, ela também pode ser usada para reduzir a quantidade de recursos utilizados pela aplicação.

Utilizando-se o ProGuard 3.7 [Lafortune 2007], o código de Stratego (já com imagens otimizadas) foi obscurecido. A Tabela 2 ilustra o tamanho do arquivo resultante do empacotamento do jogo e, também, a utilização de memória durante a execução, antes e depois do obscurecimento do código-fonte.

Tabela 2: Informações do jogo sobre obscurecimento do código-fonte (resultados em bytes)

Código obscur	Tam. JAR	Mem. max. cliente	Mem. max. servidor
Não	92.434	191.972	192.860
Sim	90.230	183.688	183.372

Depois que o código-fonte da aplicação foi obscurecido, o arquivo JAR gerado teve o seu tamanho reduzido em 2,78%. Em relação à utilização de memória durante a execução do jogo, o pico máximo de consumo medido antes do obscurecimento sofreu uma redução de aproximadamente 4,60% com a utilização da técnica. Comparando-se os dados da aplicação em seu pior caso (imagens não otimizadas e código-fonte não obscurecido) com o melhor caso (imagens otimizadas e código-fonte obscurecido), a redução do arquivo JAR resultante chegou a 75,76%; a redução do uso de memória durante a execução do jogo chegou a 73,77%.

5.3 Eliminação de métodos get/set

Para buscar um menor consumo de memória para a execução do jogo, decidiu-se não utilizar métodos `get/set` nas classes de Stratego, pois a utilização de tais métodos tende a aumentar o tamanho do *heap* de execução e o *overhead* de processamento das classes [Fasterj.com 2007]. Para atenuar esses efeitos ou eliminá-los completamente, todas as classes de Stratego foram implementadas com propriedades públicas, sem utilização de métodos `set` para sua alteração.

Para analisar os resultados dessa decisão de projeto, foram realizadas medições no tamanho do arquivo JAR resultante e na quantidade de memória utilizada para a execução do jogo, tanto na forma de atuação cliente como na forma de atuação servidor. A Tabela 3 mostra dados referentes ao jogo com classes utilizando

apenas propriedades públicas, sem uso de métodos `get/set`. Em contrapartida, a Tabela 4 mostra os mesmos atributos medidos na Tabela 3, porém com classes utilizando apenas propriedades privadas (propriedades estáticas foram deixadas públicas), com métodos `get/set` para sua manipulação.

Tabela 3: Informações do jogo não utilizando métodos get/set (resultados em bytes)

Código obscur.	Img. otim.	Tam. JAR	Mem. max. cliente	Mem. max. servidor
Não	Não	372.308	700.104	699.476
Nao	Sim	92.434	191.972	192.860
Sim	Não	370.104	694.548	692968
Sim	Sim	90.230	183.688	183.372

Tabela 4: Informações do jogo utilizando métodos get/set (resultados em bytes)

Código obscur.	Img. otim.	Tam. JAR	Mem. max. cliente	Mem. max servidor
Não	Não	374.410	691.392	691.804
Não	Sim	94.536	181904	183.484
Sim	Não	370.711	700.240	700.688
Sim	Sim	90.837	191.232	191.508

Para uma melhor compreensão dos dados obtidos, a comparação entre as Tabelas 3 e 4 está explicitada na Tabela 5. Essa tabela foi gerada subtraindo os campos da Tabela 4 com os campos da Tabela 3. Dessa forma, resultados positivos encontrados na Tabela 5 indicam que o valor do campo em questão aumentou da primeira para a segunda tabela, ao passo que resultados negativos indicam uma redução no valor.

Tabela 5: Subtração dos dados da Tabela 4 pelos dados da Tabela 3 (resultados em bytes)

Código obscur.	Img. otim.	Tam. JAR	Mem. max cliente	Mem. max. servidor
Não	Não	2.102	-8.712	-7.672
Não	Sim	2.102	-10.068	-9.376
Sim	Não	607	5.692	7.720
Sim	Sim	607	7.544	8.136

Todas as porcentagens indicadas a seguir são em relação ao tamanho da aplicação sem utilização de métodos `get/set`. Conforme mostrado na Tabela 5 e partindo da análise dos casos em que as imagens da aplicação foram otimizadas, nota-se um aumento no tamanho do arquivo JAR resultante. A segunda linha da Tabela 5 mostra um aumento de aproximadamente 2,27% no tamanho do arquivo JAR resultante, da versão sem métodos `get/set` em relação à sua equivalente, com tais métodos. A quarta linha da Tabela 5 também mostra um aumento do JAR resultante em aproximadamente 0,67%.

Em relação à utilização de memória durante a execução da aplicação, a porcentagem de variação é maior. A análise da primeira e segunda linha da Tabela 5 mostra que, para as versões do código que não foram obscurecidas, a inclusão de métodos `get/set` reduziu o consumo de memória durante a execução. A redução mais significativa ocorreu no caso do código-fonte não obscurecido e imagens otimizadas, chegando a uma queda no uso de memória de aproximadamente 5%.

Com relação ao obscurecimento de código, que reduziu o consumo de memória da aplicação em sua versão sem os métodos `get/set`, conforme mostrado na subseção 5.2, o mesmo resultado não foi obtido com a adição dos métodos `get/set`. A terceira linha da Tabela 5 mostra um aumento no consumo de memória de aproximadamente 0,96% no código de Stratego alterado (obscurecido, sem imagens otimizadas, com `get/set`) em relação ao código de Stratego original (obscurecido,sem imagens otimizadas, sem `get/set`). A quarta linha da Tabela 5, que compara a versão com código-fonte obscurecido e imagens otimizadas, mostra um aumento mais significativo do consumo de memória. De acordo com os dados obtidos, a versão alterada da aplicação (obscurecido, imagens otimizadas, com `get/set`) apresentou um aumento no consumo de memória de aproximadamente 4,30% em relação ao seu equivalente original (obscurecido, imagens otimizadas, sem `get/set`).

6 Vantagens e desvantagens

Estão expostas a seguir as principais vantagens e desvantagens, *até onde foi possível constatar*, das funcionalidades da plataforma J2ME, almejando-se o desenvolvimento de jogos de tabuleiro, baseados em turno e multiplayer (que é o caso de Stratego). Para avaliar se uma característica é uma vantagem ou desvantagem, consideraram-se elementos como:

- Facilidade de customização de elementos: programador poder decidir exatamente a localização de um elemento na tela, por exemplo;
- Facilidade para execução de tarefas ligadas à aplicação: desenhar imagens na tela, obter dados através de elementos de interface, etc;
- Recursos da linguagem: ferramentas e artifícios que a linguagem de programação apresenta ao programador para facilitar no desenvolvimento do jogo, como estruturas de dados (matriz, vetor, *buffers*);
- Qualidades da API: uniformidade e clareza nos métodos/funções da API, a fim de tornar a lógica da aplicação simples e sem obscuridades;

6.1 Vantagens

Umas das principais vantagens do J2ME é a existência de um pacote de classes voltado exclusivamente para o desenvolvimento de material para jogos *wireless*: o `javax.microedition.lcdui.game`. As classes desse pacote são estruturadas e implementadas para permitirem o uso de código nativo, aceleração de *hardware* e formatos de dados específicos do dispositivo, o que aumenta o desempenho da aplicação e reduz o seu tamanho [SUN MICROSYSTEMS, INC. 2006]. Através do `javax.microedition.lcdui.game`, o programador possui à disposição classes com funcionalidades destinadas à tarefas comuns no desenvolvimento de jogos, como consultar o estado de uma tecla em um determinado instante, possibilidade de criar camadas para a apresentação de conteúdo, criação e gerência de animações (baseadas na renderização de uma seqüência de imagens) e gerenciamento e criação de grandes áreas de conteúdo gráfico (como mapas), gerenciados através de *tiles*.

Outra vantagem da plataforma é a existência de *threads*. Embora os dispositivos móveis apresentem poucos recursos disponíveis, como baixa capacidade de processamento e pouca memória, a JVM disponibiliza ao programador o uso de *threads*. Esse recurso é útil para realizar tarefas concorrentes na aplicação, como o envio dos dados que é feito em Stratego, no qual uma *thread* envia as informações ao mesmo tempo em que a aplicação recebe e interpreta aos comandos do jogador vindos do teclado. Além de utilizar *threads* para o envio de dados, Stratego também faz uso desse recurso para desenhar gráficos: quando a tela de jogo é criada, uma *thread* é também criada e associada à tarefa, realizando as atualizações a cada ciclo do laço principal do jogo.

Uma outra vantagem encontrada é o conjunto de funcionalidades para comunicação. Conforme explanado na seção 3.3.2, o J2ME apresenta o *Generic Connection Framework (GCF)*. Através dele, o programador tem acesso a recursos de comunicação em rede ou acesso a arquivos de uma forma clara e uniforme. Estão disponíveis para uso no J2ME conexões como HTTP, HTTPS, SMS, Datagramas UDP, *sockets* e Bluetooth. Através de *sockets*, por exemplo, que são utilizados em Stratego, é possível que a aplicação atue como cliente de uma conexão ou como servidora, comunicando dados através de um protocolo próprio. Isso aumenta

as possibilidades de desenvolvimento por parte do programador, que não fica atrelado a modelos de comunicação ou protocolos pré-definidos pela plataforma, podendo escolher aquele que melhor se adapte às suas necessidades, conforme citado na seção 3.3.2.

Finalmente outra vantagem encontrada na plataforma J2ME são os vários tipos e estruturas de dados disponíveis ao programador. Existem classes que provêem uma abstração à manipulação de dados brutos, como a classe `StringBuffer`, que implementa uma seqüência variável de caracteres, permitindo a concatenação de caracteres em diferentes momentos, sem que o programador precise se preocupar com o tamanho do buffer disponível. Essa classe gerencia, também, possíveis problemas de concorrência derivados, por exemplo, da ação de múltiplas *threads* sobre um único buffer. Além dessa classe, estão disponíveis outras classes que implementam tipos de dados como `Byte`, `Double`, `Short` e `Integer`. Estão disponíveis também, vinculados a essas classes, métodos úteis ao tratamento e manipulação de dados, como o método `getBytes()`, que retorna uma seqüência de bytes que descreve uma *string*. Esse método foi utilizado em Stratego para a comunicação de dados entre os dispositivos, para que a *thread* de envio pudesse transmitir um byte da mensagem por vez, até que toda a *string* fosse completamente transmitida.

6.2 Desvantagens

A primeira desvantagem encontrada na plataforma J2ME durante o desenvolvimento de Stratego foi a forma como gráficos e eventos de teclado estão, de certa forma, atrelados. Para que uma classe seja hábil a ser desenhada na tela do dispositivo, ela precisa estender a classe `Displayable`. Para a implementação de Stratego, utilizou-se a classe `Canvas`, que estende `Displayable`. Conforme [SUN MICROSYSTEMS, INC. 2006], `Canvas` é a classe base para o desenvolvimento de aplicações que manipulam eventos de baixo nível e utilizam chamadas para desenhar gráficos na tela. É através dos métodos da classe `Canvas` que o programador pode manipular ações de jogos e eventos de teclas.

A combinação de eventos relacionados a gráficos e manipulação de teclado associadas a uma mesma hierarquia de classes torna a lógica da aplicação confusa do ponto de vista técnico. Em Stratego, por exemplo, todas as ações relacionadas a renderização de elementos na tela foram limitados à classe `GameScreen` (estende `Canvas`) que, por sua vez, também recebe e repassa os eventos de teclado realizados pelo jogador. Conforme descrito ao longo desse trabalho, `GameScreen` é a única classe autorizada a desenhar na tela, tendo, para isso, o mínimo de lógica possível, para que alterações na classe não causem efeitos sobre a lógica de outras classes. Isso, porém, não é plenamente alcançado, uma vez que `GameScreen` deve manter uma parcela de lógica da aplicação para poder manipular eventos de teclado e, então, os repassar aos cuidados de `GameBoard`. No contexto de Stratego, faz mais sentido que `GameBoard` ou `StrategoEngine` recebam eventos de teclado, porém `GameScreen` é quem realiza essa tarefa.

Outra desvantagem encontrada na plataforma foi a baixa abstração para a criação de animações. Para que o programador crie animações, é possível utilizar a classe `Sprite`, integrante do pacote de classes destinadas ao desenvolvimento de jogos. Essa classe renderiza os vários quadros encontrados em uma única imagem, de forma a criar uma animação. Os quadros na imagem são definidos pela sua divisão em células de tamanho igual, na qual cada célula resultante será interpretada como um quadro da animação. A largura e a altura de cada célula é definida pelo programador. Uma vez instanciado, o objeto `Sprite` pode ser renderizado. Para que o próximo quadro seja renderizado, é necessário que o programador chame explicitamente um método que indica que o quadro atual da animação deve ser incrementado.

Para que alguns quadros da animação sejam renderizados por mais tempo, como no caso do programador desejar uma parada na animação, é possível dizer ao objeto `Sprite` qual a ordem dos quadros a ser renderizada, através de um vetor. Nesse vetor, o programador pode fazer com o que o quadro da parada seja renderizado três vezes, por exemplo, para só então o próximo quadro ser renderizado. Um exemplo para ilustrar esse processo seria a utilização do vetor `0,1,1,1,2,3` para guiar a animação; nesse exemplo, a animação iniciaria no quadro `0` da imagem, passando depois para o quadro `1` (três vezes), em seguida para o quadro `2` e, finalmente, para o quadro `3`. Isso dará ao usuário a sensação de que a animação ficou pausada por alguns instantes no quadro `1`. Até onde foi possível constatar, a plataforma J2ME, em si, não apresenta ferramentas que auxiliem o programador na criação de animações de uma forma sistemática e automatizada. Isso pode vir a aumentar o tempo de produção de um jogo em virtude da elaboração de animações com pouco nível de abstração do processo, conforme descrito. A seção 7 mostra outras plataformas de desenvolvimento com recursos mais aprimorados em relação à criação e gerência de animações.

Outra desvantagem encontrada no desenvolvimento de jogos ao estilo Stratego é a forma como elementos de GUI (*Graphic User Interface*), ou elementos gráficos, são tratados. Conforme citado anteriormente, para que um elemento seja mostrado na tela do dispositivo, ele precisa estender `Displayable`. Qualquer elemento gráfico que estenda essa classe pode, então, ser desenhado na tela. A plataforma J2ME apresenta vários elementos de GUI para auxiliar no desenvolvimento de aplicações, como a classe `Alert`, que mostra um aviso na tela, e a classe `ChoiceGroup`, que mostra um grupo de opções de escolha. Se o programador precisar mostrar vários desses elementos na tela, eles devem, então, ser agregados a um objeto da classe `Form` (que estende `Displayable`), para que sejam desenhados todos juntos. A forma de desenho na tela é ditada pelo algoritmo de layout da classe `Form` [SUN MICROSYSTEMS, INC. 2006].

Essa estruturação dos elementos, durante o desenvolvimento de Stratego, trouxe algumas complicações ao código relacionado à interface com o usuário. Quando o jogo se inicia, a classe `GameScreen` é setada como sendo o elemento que deve ser desenhado na tela. Nesse momento, a aplicação tem total controle sobre aquilo que é apresentado ao usuário, como a imagem de fundo, a imagem das peças do jogo, etc. Em determinados momentos, é necessário mostrar ao jogador elementos de interface, como grupos de escolha ou campos de texto. Como `GameScreen` é a classe corrente que está sendo desenhada, não é possível que outros elementos (como o `ChoiceGroup`) sejam mostrados sem que `GameScreen` interrompa suas tarefas.

Para que o `ChoiceGroup` seja desenhado, uma instância da classe `Form` deve ser setada como o elemento que deve ser desenhado na tela (`focus` de desenho), o que faz com que `GameScreen` interrompa seus desenhos. Para desenhar elementos de interface sobre a tela de jogo, o programador pode, por exemplo, 1) abrir mão da tela de jogo e, então, instanciar um novo `Form`, agregar elementos de GUI a ele e instruir o dispositivo a desenhar o form (fazendo com que o jogador não veja mais qualquer elemento do jogo, como o mapa de combate, no caso de Stratego); ou 2) implementar seus próprios componentes de GUI, a fim de que esses possam ser desenhados pela classe que está desenhando no momento (`GameScreen`, no caso de Stratego). Há, porém, alguns elementos de interface que podem ser mostrados no momento que `GameScreen` está desenhando, como um objeto da classe `Alert`. Quando instanciado, o programador indica ao construtor dessa classe qual elemento deve passar a ser desenhado depois que o aviso desaparecer.

Mesmo com essa abordagem, `GameScreen` cede a tarefa de renderizar todos os elementos da tela para que o objeto da classe `Alert` possa ser desenhado. Nesse momento, não haverá mais elementos do jogo na tela, apenas o aviso (se o programador desejar que uma parcela do jogo continue sendo mostrada, ele deverá implementar seu próprio aviso, como no caso do `ChoiceGroup`).

7 Paralelo com outras plataformas de desenvolvimento

A fim de complementar a análise da plataforma J2ME e abordar de forma mais técnica os tópicos levantados na seção 6, a seguir apresenta-se um paralelo entre as funcionalidades da plataforma utilizada para o desenvolvimento de Stratego com as

funcionalidades semelhantes encontradas em outras plataformas. Não é o objetivo dessa seção prover uma análise profunda das plataformas citadas, mas sim enriquecer a análise das vantagens e desvantagens do J2ME, citadas anteriormente.

7.1 Paralelo entre as vantagens

Uma das vantagens citadas da plataforma J2ME é a existência de um pacote destinado exclusivamente ao desenvolvimento de jogos. Comparando-se com a plataforma Flash Lite, em sua versão 2.0, até onde foi possível constatar, não existe um pacote de classes explicitamente declarado para o desenvolvimento de jogos. Embora a plataforma Flash Lite apresente vários recursos gráficos aprimorados, que serão citados à frente, não há a centralização desses em um pacote explícito como aquele visto no J2ME. Conforme já descrito, o pacote `javax.microedition.lcdui.game` utiliza classes que são implementadas e desenhadas para tirarem vantagem do *hardware* do dispositivo, aumentando o desempenho dos processos relacionados.

Outra vantagem citada anteriormente foi a existência de *threads* em J2ME, que aumentam a capacidade da aplicação em executar tarefas de forma concorrente. Até onde foi possível constatar, na plataforma Flash Lite não está disponível ao programador a criação e utilização de *threads*. É possível, porém, simular a execução concorrente de operações através do uso da função `setInterval()`, no qual o programador define um método de uma classe para ser executado repetidamente, com um intervalo customizável entre as chamadas. Através das propriedades da classe, é possível manter estados do processo, simulando a preservação de contexto de *threads*. Essa abordagem, porém, é limitada, uma vez que programador não detém um fluxo exclusivo de execução, como aquele proporcionado pelas *threads* do J2ME. Conforme testes realizados, métodos que sejam chamados por `setInterval()` não podem conter laços infinitos, como um `while(true)`, uma vez que eles priorizam a execução somente desse método, pois ele não retorna, fazendo com que `setInterval()` aguarde sua finalização indeterminadamente, o que faz com que a aplicação não responda mais (resultando em um *crash*). Em relação à plataforma BREW, a partir do SDK 3.0, está disponível ao programador recursos como a `IThread`, que implementam a execução de tarefas de forma paralela [QUALCOMM INCORPORATED 2006].

Uma vantagem da plataforma J2ME também citada é o grande conjunto de recursos para comunicação em rede. Em um paralelo com a plataforma Flash Lite, constatou-se que essa apresenta meios para troca de informações em rede limitados a protocolos pré-definidos, em sua grande maioria. A plataforma possibilita ao programador o carregamento de arquivos externos (como documentos de texto ou arquivos XML) através do protocolo HTTP (se o arquivo for remoto), utilização de SMS e envio de um conjunto de informações (como campos de um formulário) para um servidor, utilizando HTTP para a comunicação [ADOBE SYSTEMS INCORPORATED 2006a]. No conjunto de meios de comunicação com protocolos mais flexíveis (com características definidas pelo próprio programador, por exemplo), está disponível a classe `XMLSocket`, que permite ao programador abrir um canal de dados com algum servidor e, através de XML, trocar informações. Há várias restrições a serem consideradas nesse processo, como a porta na qual o *socket* é aberto, que deve ser, obrigatoriamente, maior que 1024; e que cada mensagem enviada é um documento XML terminado por um byte zero (o servidor que receber a mensagem deve entender essa convenção). Em comparação com as funcionalidades do *Generic Connection Framework* do J2ME, que apresenta uma API uniforme e clara para comunicação em rede, através de vários protocolos e formas de transmissão de dados, as possibilidades de comunicação da plataforma Flash Lite são mais limitadas.

Em um paralelo com a plataforma BREW, é possível encontrar algumas semelhanças em termos de funcionalidades de comunicação. BREW disponibiliza bibliotecas como `ITAPI`, que possibilitam acesso a serviços de voz e SMS, `ISocket` que permite a manipulação de conexões tanto TCP quanto UDP, `IWeb` que facilita a condução de transações web sobre vários protocolos e `IPort` que disponibiliza uma comunicação genérica com várias interfaces.

Tabela 6: Resumo da comparação das vantagens entre as plataformas J2ME, BREW e Flash Lite

Elemento analisado	J2ME	BREW	Flash Lite
Existência de pacote com classes voltadas ao desenvolvimento. de jogos	Sim	Não	Não
Existência de *threads*	Sim	Sim, a partir do SDK 3.0	Não
Meios de comunicação em rede	Grande conjunto através do GCF: HTTP, Bluetooth, HTTPS, *socket*, UPDDatagram, SMS e comunicação serial	Conjunto considerável: HTTP, *socket*, HTTPS, SMS, comunicação serial.	Conjunto reduzido: HTTP, SMS e XMLSocket (*socket* com diversas limitações).
Linguagem de programação fortemente tipada	Sim	Sim	Não
Tipos nativos de dados	Grande conjunto, como `int`, `byte`, `char` e `short`.	Os mesmos tipos de dados nativos da linguagem C, como `int`, `float`, `char`, `long` e `short`.	Inexistentes. A plataforma utiliza apenas classes para representar dados.
Classes e estruturas de dados	Grande conjunto de classes úteis já implementadas, como `StringBuffer` `String`, `Float`, `Vector` e `Hashtable`	*(Não analisado)*	Conjunto considerável de classes, como `String`, `Array`, `Boolean` e `Number`.

Outra vantagem citada sobre o J2ME são os vários tipos e estruturas de dados disponíveis ao programador. Conforme já explanado, J2ME apresenta um grande conjunto de tipos nativos de dados, como `int`, `char` e `byte`, além de classes que implementam estruturas de dados úteis à programação, como `StringBuffer`. Em um paralelo com a plataforma BREW, um grande conjunto de tipos de dados também é fornecido, assim como aqueles que são encontrados na linguagem de programação C++ [QUALCOMM INCORPORATED 2006]. Em ambas as plataformas, a linguagem de programação apresenta tipagem forte, o que evita que uma variável tenha seu tipo alterado ao longo do tempo.

Na plataforma Flash Lite, até onde foi possível constatar, existem também estruturas de dados já implementadas úteis à programação, como `Array`, `String` e `Boolean`, porém a linguagem de programação é fracamente tipada e não apresenta distinção entre certos tipos nativos de dados. Em relação à tipagem, conforme testes realizados, é facultativa a definição do tipo de uma variável, o que permite que uma variável tenha o seu tipo alterado ao longo do tempo. Se o programador explicitamente especificar o tipo de uma variável, forçando uma tipagem forte, o compilador acusa erro de incompatibilidade de tipos se houver a tentativa de utilização de algum tipo não compatível. Em relação à distinção de tipos nativos de dados, não existe distinção entre números inteiros (com sinal ou sem sinal) e números de ponto flutuante. Se o programador deseja tipar uma variável como sendo numérica, ele possui à disposição apenas o tipo `Number`, que engloba todos os tipos de dados numéricos existentes na plataforma.

A tabela 6 apresenta um resumo da comparação das vantagens da plataforma J2ME com as outras plataformas.

7.2 Paralelo entre as desvantagens

Em relação às desvantagens encontradas na plataforma J2ME, a primeira delas é a forma como a manipulação de eventos de teclado e a geração de gráficos são tratados de forma atrelada, conforme citado anteriormente. Em um paralelo com plataforma BREW, até onde foi possível constatar, a manipulação de eventos do teclado e geração de gráficos são elementos distintos tratados por interfaces diferentes. Para tratar eventos do teclado, a aplicação da plataforma BREW deve implementar o método `nomeClasse_HandleEvent()`, onde `nomeClasse` é o nome da classe que descreve a aplicação. Para desenhar na tela, a aplicação deve

fazer uso, por exemplo, de `IDISPLAY_Update()`. Em relação à plataforma Flash Lite, conforme testes realizados e até onde foi possível constatar, não está ao alcance do programador a realização do controle da renderização na tela do dispositivo; essa tarefa é feita de forma transparente pelo *Flash Player* do dispositivo móvel. Em relação à manipulação de eventos do teclado, o programador deve criar e vincular um objeto a um *listener* de teclado, através da chamada de `Key.addListener()`. O *listener* informa seus ouvintes cada vez que o *status* do teclado é alterado, como no caso de uma tecla ter sido pressionada. Tanto quanto na plataforma BREW quanto na plataforma Flash Lite, não há um forte atrelamento entre manipulação de eventos do teclado e geração de gráficos, assim como é feito em J2ME.

Outra desvantagem encontrada no J2ME foi o baixo nível de abstração para a criação e gerência de animações, o que pode aumentar o tempo gasto para o desenvolvimento de um jogo. Conforme já explanado, J2ME utiliza a classe `Sprite` para renderizar os vários quadros de uma imagem composta, dando ao usuário a noção de movimento. Na plataforma BREW, o programador pode fazer uso da ferramenta *BCI Authoring*, que possibilita a conversão de imagens para o formato BCI (*BREW Compressed Image*) e, também, a criação de animações. Através dessa ferramenta, o programador pode abrir várias imagens e, então, criar uma animação (as imagens são empacotadas, criando a animação, na ordem em que foram abertas). O programador pode customizar o intervalo entre um quadro e outro da animação, ainda fazendo uso da ferramenta *BCI Authoring*.

Na plataforma Flash Lite, conforme testes realizados, é possível criar animações através de códigos ou através da IDE de programação da plataforma. Utilizando a IDE, o programador pode definir quadros-chave da animação, criar interpolações de movimento entre eles e definir o tempo em que um quadro é mostrado, por exemplo. Através da interpolação, a IDE da plataforma baseia-se em dois quadros-chave informados pelo programador, criando os quadros intermediários para que o primeiro quadro-chave seja animado no segundo quadro-chave. As animações são representadas pela classe `MovieClip`, que apresenta métodos intuitivos para controle do fluxo da animação, como `play()`, `stop()`, `gotoAndPlay()`, `gotoAndStop()`, `prevFrame()` e `nextFrame()`. Depois de iniciada com `play()`, a animação é renderizada até o seu final sem interferência do programador.

Finalmente outra desvantagem encontrada na plataforma J2ME foi como elementos de GUI (*Graphic User Interface*), ou elementos gráficos, são tratados. Conforme explanado, em J2ME, para que um elemento seja desenhado na tela, ele deve estender alguma classe que estenda `Displayable` e, também, deve ganhar o *focus* de desenho. Em Stratego, não foi possível mostrar elementos de GUI ao mesmo tempo que a tela de jogo era mostrada, porque somente um elemento por vez poderia possuir o *focus* de desenho (eles não poderiam ser desenhados ao mesmo tempo). Em um paralelo com a plataforma Flash Lite, tanto elementos de GUI quanto outros elementos (como animações) são desenhados de forma diferenciada. Conforme testes realizados, o `xFlash Player` do dispositivo realiza o desenho dos elementos que estão no palco (área visível do *display*). O programador pode utilizar a IDE de programação da plataforma para inserir elementos gráficos manualmente no palco, dispondo-os conforme suas necessidades, ignorando preocupações relacionadas ao *focus* de desenho, como no caso da plataforma J2ME. O programador pode, também, inserir elementos gráficos em tempo de execução, contanto que esses elementos estendam `MovieClip`. Uma vez adicionado ao palco, um elemento pode ser posicionado, rotacionado ou ter suas proporções alteradas, afim de atender às necessidades do programador. Através dessa estrutura, o programador pode colocar elementos na tela em tempo de execução, como avisos ou caixas de texto, sem que a tela de jogo deixe de ser desenhada ou que o jogador perca contato visual com ela. Há, também, vários elementos de GUI já implementados e disponíveis para uso na IDE de programação da plataforma Flash Lite, como `ChoiceGroups` e `Alerts`, o que não obriga o programador a desenvolver seus próprios elementos de interface para que eles possam ser mostrados ao mesmo tempo que a tela de jogo.

A tabela 7 apresenta um resumo da comparação das desvantagens da plataforma J2ME com as outras plataformas.

Tabela 7: Resumo da comparação das desvantagens entre as plataformas J2ME, BREW e Flash Lite

Elemento analisado	J2ME	BREW	Flash Lite
Manipulação de eventos do teclado	Métodos da classe `Canvas`	Através do método `nomeClasse_ HandleEvent()`	Através de um *listener* de teclado
Desenhos no display	Métodos da classe `Canvas`	Método `IDISPLAY_ Update()`	Feita de forma transparente pelo *Flash Player*. O programador não tem meios para interferir no processo.
Gerenciamento e criação de animações	Manipulação de baixo nível, através da classe `Sprite`. A plataforma, por si só, não possui ferramentas para criação de animações.	Disponibilizado através da ferramenta BCI Authoring, da própria plataforma.	Manipulação de alto nível, através da classe `MovieClip`. Criação de animações através da IDE do Macromedia Flash.
Desenho de elementos de interface (GUI)	*Focus* de desenho exclusivo: elementos GUI da plataforma não podem ser desenhados ao mesmo tempo que telas de jogo criadas pelo programado (se a tela de jogo não for um `Form`)	*(Não analisado)*	Não há *focus* de desenho. Elementos GUI da plataforma podem ser desenhados ao mesmo tempo que as telas de jogo.

8 Conclusão

Este artigo apresentou uma exposição das funcionalidades das plataformas J2ME, BREW e Flash Lite, com maior ênfase e aprofundamento na primeira. Através de um estudo de caso, o jogo multi-jogador Stratego, utilizando-se a plataforma J2ME, mostrou-se as vantagens, desvantagens e peculiaridades da plataforma no que se refere ao desenvolvimento de jogos para dispositivos móveis. Com base nas experiências vivenciadas durante a implementação do caso de uso, traçou-se um paralelo das vantagens e desvantagens do J2ME com outras duas plataformas, a BREW e Flash Lite, a fim de enriquecer a análise e prover subsídios a outros desenvolvedores sobre a criação de jogos através da utilização da plataforma J2ME.

Outro assunto abordado foi o levantamento e execução de técnicas ligadas à redução do consumo de recursos por parte de jogos para dispositivos móveis, a fim de atenuar o problema de desenvolver aplicações para ambientes com pouca memória e baixo processamento. Realizou-se otimização de imagens, obscurecimento de código e redução do uso de um número elevado de classes e de métodos `get/set`. A execução dessas técnicas na implementação de Stratego possibilitou a coleta de informações importantes, como a quantidade de memória consumida por classes que utilizavam métodos `get/set` e o consumo de classes que não utilizavam tais métodos, através de tabelas de dados e comparações. Tais dados podem ser utilizados, por exemplo, como base para analisar a técnica de otimização utilizada, analisar a plataforma e suas funcionalidades ou analisar o impacto no consumo de recursos em aplicações semelhantes a Stratego.

Como o principal foco do trabalho estava na análise de assuntos relevantes ligados à plataforma a partir de um caso prático de uso, tópicos relacionados ao jogo Stratego foram deixados de fora do escopo de implementação e análise. Nesse sentido, podem-se desenvolver trabalhos futuros em relação a esses tópicos não contemplados, como aperfeiçoamento da interface do jogo, através do armazenamento da disposição de peças, a fim de evitar que o jogador tenha que posicionar todas as peças a cada partida (ele pode simplesmente escolher uma organização que já tenha feito e armazenado).

Outro trabalho possível é a utilização de diferentes formas de comunicação entre os dispositivos, como Bluetooth ou HTTP, a fim de analisar seus problemas e vantagens de forma prática. Embora a análise dos recursos de comunicação disponíveis no J2ME tenham sido cobertos pelo presente trabalho, não de uma forma mais profunda em virtude da cobertura de outros assuntos, a utilização

de outros meios de comunicação poderá disponibilizar informações úteis sobre esse tópico crítico no desenvolvimento de jogos, que é a comunicação entre os dispositivos.

Outra possibilidade seria a implementação do jogo Stratego em uma das outras duas plataformas de desenvolvimento citadas, BREW e Flash Lite. Através dessa implementação, a análise das vantagens e desvantagens da plataforma J2ME seria maximizada e aumentaria os subsídios informacionais gerados, o que poderia contribuir com estudos e análises futuras entre as plataformas abordadas.

Referências

ADOBE SYSTEMS INCORPORATED, 2006. Adobe flash lite. Disponível em: <http://www.adobe.com/products/flashlite/>.

ADOBE SYSTEMS INCORPORATED, 2006. Adobe flash player. Disponível em: <http://www.adobe.com/products/flashplayer/>.

ADOBE SYSTEMS INCORPORATED, 2007. Adobe flash lite 2.1 datasheet. Disponível em: <http://www.adobe.com/products/flashlite/productinfo/-overview/datasheet.pdf>.

ARDFRY IMAGING, LLC, 2007. Pngout. Disponível em: <http://www.ardfry.com/pngoutwin/>.

BARROS, A. R. 2005. Estudo sobre a utilização da tecnologia wireless no desenvolvimento de aplicações distribuídas multi-usuário. Tech. rep., Universidade Federal de Lavras, Lavras.

FASTERJ.COM, 2007. Java performance tuning.

HEDLUNG, M., AND MAXE, R. 2004. *Multiplayer Games in Mobile Terminals*. Master's thesis, Lulea University of Technology, Sweden.

KARLSSON, B. F. F., NASCIMENTO, I. F., BARROS, T. G. F., SANTOS, A. L. M., AND RAMALHO, G. L. 2003. Análise de tecnologias de desenvolvimento de jogos para dispositivos móveis. In *Workshop Brasileiro de Jogos e Entretenimento Digital (WJOGOS)*, SBC.

KLINGSHEIM, A. N. 2004. *J2ME Bluetooth Programming*. Master's thesis, University of Bergen, Norway.

LAFORTUNE, E., 2007. Proguard. Disponível em: <http://proguard.sourceforge.net/>.

LINSALATA, D., AND SLAWSBY, A., 2005. Addressing growing handset complexity with software solutions. Disponível em: <http://www.adobe.com/mobile/news_reviews/-articles/2005/idc_whitepaper.pdf>.

MAHMOUD, Q. H., 2003. J2me low-level network programming with midp 2.0, April. Disponível em: <http://developers.sun.com/techtopics/mobility/midp/-articles/midp2network/>.

NOKIA CORPORATION. 2003. *Developer Platform 1.0 for Series 40 and Series 60: Known Issues. Version 1.1*, Jun.

NOKIA CORPORATION, 2006. Ngage.com. Disponível em: <http://www.ngage.com>.

ORTIZ, C. E., 2003. The generic connection framework, Aug. Disponível em: <http://developers.sun.com/techtopics/mobility/midp/-articles/genericframework/>.

PEREIRA, D. A. 2006. Jogos ubíquos com bluetooth. Tech. rep., Universidade Federal de Pernambuco, Recife.

QUALCOMM INCORPORATED. 2006. *Qualcomm Brew Developer Home*. Disponível em: <http://brew.qualcomm.com/brew/en/developer/overview.html>.

RABELO, S., HAHN, R. M., AND BARBOSA, J. 2005. Bluegame: Um jogo móvel multijogador baseado em comunicação bluetooth. In *Workshop Brasileiro de Jogos e Entretenimento Digital (WJOGOS)*, no. 4, SBC.

SACRAMENTO, D. A. A., ROCHA, G., AND ESMIN, A. A. A. 2005. Proposta de um framework para construção de jogos multi-usuários para telefones celulares. In *Workshop Brasileiro de Jogos e Entretenimento Digital (WJOGOS)*, SBC.

SUN MICROSYSTEMS, INC. 2006. *OverView (MID Profile)*, Nov.

SUN MICROSYSTEMS. 2006. *The Java ME Platform*. Disponível em: <http://java.sun.com/j2me/>.

TRUTA, C., 2007. Optipng. Disponível em: <http://optipng.sourceforge.net/>.

WIKIPEDIA, 2006. Stratego. Disponível em: <http://en.wikipedia.org/wiki/Stratego>.

WISNIEWSKI, D., ET AL. 2005. 2005 Mobile Games White Paper. In *Game Developers Conference*. Disponível em: <http://www.igda.org/online/>.

Implementation of a Service Platform for Crossmedia Games

Fernando Trinta
University of Fortaleza

Davi Pedrosa, Carlos Ferraz, Geber Ramalho
Federal University of Pernambuco

Abstract

Crossmedia games are a genre of pervasive gaming where a single game instance can be played with a variety of heterogeneous devices that support different forms of players' participation and deliver different game experiences. In this document, we present the PM2G initiative, a Service-Oriented Architecture aiming at support crossmedia games' development and execution. Due to their relevance in this paper, Content Adaptation and Interaction Adaptation Services are detailed. We also present a case study, a game called Pervasive Wizards, which is used to validate PM2G architecture. Finally, we also present some performance results obtained in our experiments.

Keywords:: Crossmedia games, Middleware

Author's Contact:

trinta@unifor.br
{dvp, cagf, glr}@cin.ufpe.br

1 Introduction

According to the IPerg Project (EU Integrated Project on Pervasive Games), a crossmedia game is the one in which the content of a single game instance is made available to its players through different access interfaces [Lindt et al. 2007]. Each player experience differs by combining those interfaces that vary from desktop PCs to augmented reality systems and even cell phones.

Crossmedia games reflect a view for future digital games in which these applications will use technological advances in several areas such as mobile and context-aware computing and large scale distributed systems. Within this scenario, games will assume the role of persistent virtual worlds where players will have potential access to them, anytime, anywhere, using heterogeneous devices and communication networks.

A crossmedia game feasibility involves a spectrum of challenges that vary from technical problems, such as support and integration of different technologies to non technical-issues, such as a suitable game design for these applications. Several research groups started to propose solutions for different issues regarding the construction of crossmedia games. One of these solutions is the PM2G proposal (Pervasive Multiplayer Multiplatform Game)[Trinta et al. 2006a][Trinta et al. 2007]. This proposal focuses on the support for Crossmedia Role Playing Games (RPG).

The purpose of PM2G, in particular, aims to facilitate crossmedia games development and execution through the specification of a set of services in a middleware layer. This layer deals with specific aspects of these applications, including scenarios, application and usage models, as well as the initial design of its architecture.

This paper presents results from a case study using the PM2G architecture. For this purpose, it was designed and implemented a digital RPG named Pervasive Wizards, in accordance with the PM2G application model. Through this case study, we were able to evaluate PM2G architecture with regards to the performance of the main proposed services.

This article is organized in seven sections. Following this introduction, the second section reviews related works to PM2G initiative. The third sections summarizes the main PM2G concepts that are relevant to this document. The fourth section introduces the PM2G architecture. The fifth section highlights the services evaluated in our experiments. The following sections present our case study and the results from our experiment. Finally, the last section concludes this document.

2 Related Work

To the best of our knowledge, few studies and projects have been published concerning crossmedia games. Most of them make an overview of advantages achieved by incorporating mobile devices into massively multiplayer games (MMGs), but they do not show concrete and effective solutions to realize this scenario.

Fox [Fox 2003] was the first to propose the ideia of multiplatform games, where the author highlights needs for adapting the user playability when using mobile devices, especially mobile phones. Changes in this playbility would affect not only how a player perceives the game, but also how each player interacts with the virtual world. Fox presents some interesting ideas to enhance players experience using specific mobile computing services, and some of his ideas are used in the PM2G proposal.

Han et al [Han et al. 2004] in a short paper presents a situation-aware middleware for multiplatform games. This middleware supports five different platforms: arcade machines, consoles, desktop computers, PDAs and mobile phones. Using their middleware, players may migrate their game session from one platform to another, according to their context. Their architecture uses sensors that detect situations (such as the presence of a more powerful device nearby a player), and associate a specific behavior with each situation (for instance, transfer of the game state between devices). However, the lack of more detailed information about their experiments do not allow us to make a deeper analysis of their work.

MMMOG Platform [Menezes et al. 2006] provides support for Mobile Massively Multiplayer Games. Their architecture offers a set of services in a middleware layer that deals with common issues of Mobile MMG's development and execution. However, MMMOG Platform focus on powerful cell phones only.

Finally, IPerg Project (http://iperg.sics.se/) is the most directly related to the PM2G initiative. IPerg aims to study innovative and attractive forms for pervasive games. This ongoing project aims to study several aspects for crossmedia games, like game design, available market and support infrastructure. IPerg proposes associations between the real world and the virtual world, through geometric transformations [Lindley 2005]. For instance, an user displacement in the real world would also change the location of his avatar in the virtual world. In the same subproject, Koivisto [Koivisto and Wenninger 2005] presents a survey that collects player's feedback about proposed features for MMGs with Mobile support, such as message notification or asynchronous playability. Some of these features are also included in PM2G proposal. With regards to the support requirements, IPerg only presents a reference architecture, and more detailed information is expected as the project is under development.

3 PM2G Initiative

The survey of related works allowed us to verify that there is not a clear vision of how multiplayer games and pervasive computing must be integrated. In such a way, it was necessary to establish a clear and concrete point of view, through scenarios that define the interactions intended in such integration. The PM2G created scenarios that pointed out possible situations for players that take part in a crossmedia game. These scenarios enforces the idea that players may use technological advances already present in their daily routine, to increase their immersion inside the game. According to the PM2G scenarios, the game embodies the role of an omnipresent pervasive application. In a brief description, a PM2G is a distributed multiplayer game that can be played in different ways by a same player, using heterogeneous devices and communications technologies. These scenarios may be found in [Trinta et al. 2006b] [Trinta et al. 2006a].

Based on these scenarios, a analysis was done by looking at games that were more suitable to these scenarios. Games may use different themes that vary from car racing, army battles and even sport simulation. Each one of these themes have peculiar requirements that makes a uniform treatment inviable through a support platform for such applications. In the proposal PM2G, the target-theme was the RPGs, mainly by the fact that these games represent 97% of online games market [Woodcock 2006]. RPGs characterize themselves by a complex narrative in which each player incorporates the role of a character, commonly called avatar. The virtual world is composed by different races (humans, elves) accomplishing missions, commonly called quests, like searching for objects and places. These games focus on the behaviour and evolution of each avatar though the accumulation of wealth and experiences acquired during missions accomplishments.

After the definition of these scenarios and the choice of the target-theme for PM2G games, two models were specified: an application model and a usage model. The former includes elements that compose a PM2G application and allow us to create a terminology for PM2G elements. The latter stands out how users must interact with the game. Figure 1 shows some concepts from the PM2G application model. The main concepts relevant for understanding this document will be presented further. Other details about the scenarios and PM2G models can be found in [Trinta et al. 2007] and [Trinta et al. 2006a].

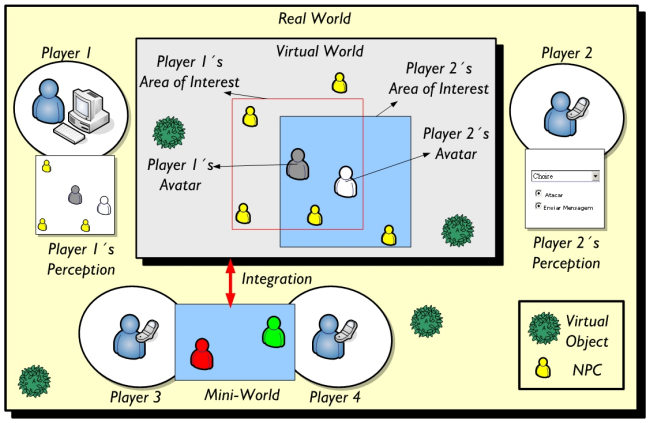


Figure 1: PM2G Application Model

In a PM2G game, each player interacts with the world through his avatar, his representation in the virtual world. This world represents the environment where the game takes place. This environment includes all players' avatars, virtual objects and NPCs (Non-Player Characters). Virtual objects represent weapons, portions or treasures that can be collected by the players according to rules and objectives defined by the game designer. Like the majority of the distributed games, the player interacts only with a part of the virtual world. This subset is called area of interest and typically includes only avatars and objects which the player may interact at a given moment.

All concepts previously presented are common to ordinary digital RPGs. The PM2G application model includes four new concepts concerned with the support for heterogeneous devices. The first concept is called *perception*. This concept indicates that a player's area of interest might have different representations, according to the player's access features at a given moment. These features create a second concept: *context*. In a pervasive application, context is concerned with any relevant information for its behaviour. In PM2G games, these information includes the device, personal preferences and location (when available) of the player when accessing the game. The relation between these concepts indicates that a player's perception is directly influenced by his context. For instance, in Figure 1, although player 1 and player 2 are topologically close to each other in the virtual world and owning elements in common, they have different perceptions of their areas of interest. Whereas the player 1 has a set of images as his perception, player 2 has his perception diminished to a textual information that describes the elements close to his avatar.

A third concept introduced by the PM2G application model is *interaction*. This concept aims to create a mechanism that facilitates the accomplishment of some tasks by mobile players, due to their inherent limitations, such as small screen or a reduced number of keys for inputting data. For instance, according figure 1, player 1 may attack player 2 using his mouse to choose an available weapon from his inventory and then click over player 2's avatar. Player 2, on the other hand, may attack player 1 after choosing him among those present in his enemies textual list, as well the weapon to be used in this attack. This case follows the idea proposed by Fox [Fox 2003], in which mobile players use high level actions, such as "Fighting", "Collecting" or "Defending an area". By choosing an action, the game simulation takes care of all necessary steps to perform the action selected by the mobile player. Player 1 would see movements and mutual attacks from both avatars, while player 2 may be notified later via a textual report describing the results of his actions.

Finally, the *mini-world* concept is related to new opportunities of interaction, especially for mobile players. Mini-world represents parallel games which their results are incorporated into the shared virtual world. They are constituted as parallel scenarios with direct interaction between layers via mobile devices, through *ad-hoc* or WLANs(Wireless Local Area Networks). These scenarios can be understood as isolated games among mobile players. However, mini-worlds have consequences in the simulated virtual world. The PM2G game designer is the one that makes these scenarios related to the central theme that conducts the virtual world. As an example, in a medieval PM2G, a mini-world might represent a duel between two players, in which the winner takes objects from the defeated player and may use them in the game.

The usage model indicates how players interact with the PM2G game. Figure 2 illustrates this model. Each player enjoys a PM2G through interaction sessions. A session is characterized by an elapsed time interval in which a player interacts with the game using a specific device. When the session ends, the player may migrate to another device and start a new session, possibly continuing the actions where the last session was concluded. Each new session requires an authentication process, where the player informs his login and password.

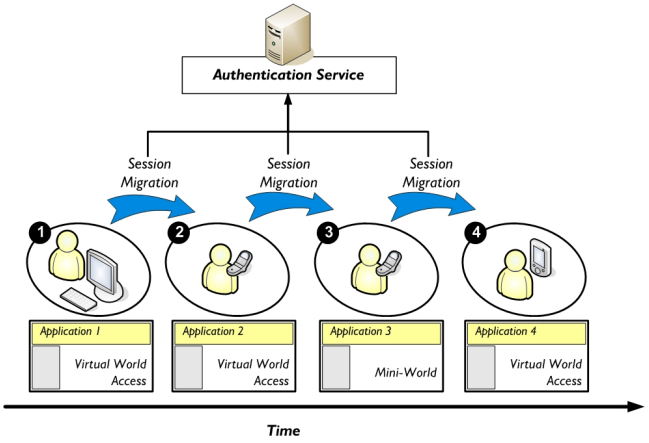


Figure 2: PM2G Usage Model

4 PM2G Middleware

In order to enable PM2G scenarios and models, our proposal is based in the traditional support model for distributed games. This model uses a layer of services through a middleware platform. Five services were designed concerned to specific PM2G concepts from our scenarios and models. Figure 3 shows the PM2G architecture.

The architecture follows the Client/Server model, in which the server includes the game simulation and the set of services responsible for different PM2G functionalities. This approach is influenced by factors such as: *(i)* replicated servers may be used to improve scalability, *(ii)* better management control over game processes,

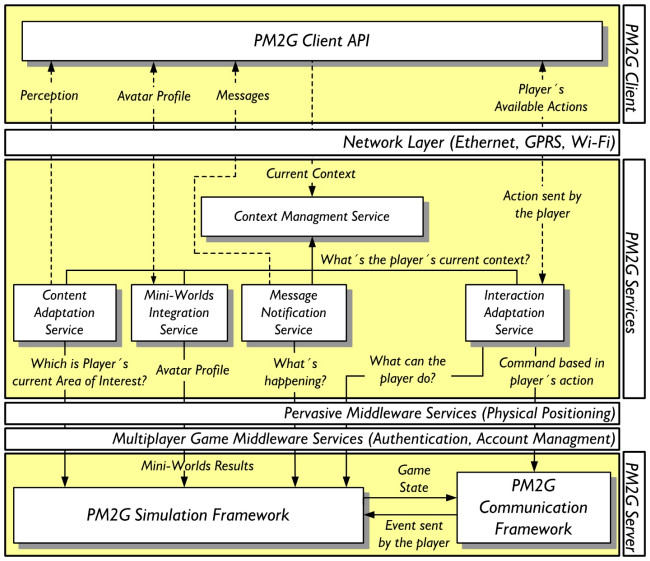


Figure 3: PM2G Architecture

(iii) higher cheating tolerance and *(iv)* easiness of implementation.

The PM2G architecture is divided in three main layers. The lower layer is referenced as the PM2G server and includes two frameworks: one responsible for the game simulation and another for dealing with the communication between players' clients and the game simulation. All the communication between players and the simulated virtual world is done via events that are received by the communication framework and sent to the simulation framework. On the other hand, the simulation framework offers to clients and services the current game state at a given moment. This state represents the set of information about each player avatar and other game elements (NPCs or objects). This set of information is then sent to the player applications which update their graphichal interface in order to visualize the game state.

The second layer presents the five PM2G services. The first one is the Context Management Service that holds contextual information about players. This contextual information includes *(i)* the device being used, *(ii)* personal preferences and possibly *(iii)* the geographic location of each player. This service has a central role in PM2G solution by the fact that contextual information guides the internal behavior of others components. Any interested service may inquires the Context Management Service about any player context, when necessary.

This is the case of the Content Adaptation Service which is responsible for sending the current player game state, in accordance with his current context. According to the application model, each player interacts only with his area of interest in the virtual world. The service adapts information about this area of interest in a more suitable format to current player's context. The results from these transformations follow the concept of perception from the PM2G application model.

A third service is the Interaction Adaptation which aims to facilitate the execution of tasks by mobile players. To achieve this objective, the service retrieves the available user's actions based on the current game state and player's context. These actions represent the high-level commands previously cited in the PM2G application model (Section 3). These commands are sent to the player that chooses which of them he wants to perform. In a reverse process, the service receives the action and transforms it in one or more conventional commands, which are sent to the communication framework, and lately to simulation framework for its execution in the virtual world.

Another PM2G service is the Messages Notification. This service aims to monitor the game simulation, searching for events that are relevant to each player. When these events occur, the service uses player context information to send messages about these events. These messages utilize different communication technologies such as SMS or e-mail, according player's current context.

The last service is the Mini-worlds Integration. This service aims to support the mini-world concept from the PM2G application model. It has as main functions: *(i)* to allow the exchange of profile information between the virtual world and mini-worlds, such as weapons ou strength level; *(ii)* to allow that results from these mini-worlds to be integrated later and used in the virtual world.

Following the middleware approach, all PM2G services are placed above existing layer services, related either with middleware platforms for pervasive computing as well as multiplayer games. This approach indicates that the PM2G architecture aims to reuse these services, such as a Physical Positioning Service.

Finally, the last layer presents the PM2G client's application. This application receives and sends information, either to the services as to the game as well, through an API (Application Programming Interface).

Among all services described in the last section, two of them are directly related to this work. They are: the Content Adaptation Service and the Interaction Adaptation Service. These services guide player's playability, the way each user plays the game. Given its importance in this work, the following subsections summarize their functionalities. The reader may find more information about them in [Pedrosa et al. 2006].

4.1 Content Adaptation Service

In this service, the content adaptation process defines the concept of *adaptation strategy*. A adaptation strategy is associated to each supported device and determines possible transformations that must be performed before any game information reachs the client application. Transformations procedures done by the Content Adaptation Service are classified in two processing types: logical procedures and presentation procedures. The former includes procedures that modify the contents of an area of interest, such as the amount of objects or their positions in the virtual world. The latter prepares the data to be sent to the client application. Adaptation strategies aims to avoid that mobile devices be responsible for any processing procedure concerned with adapting data from the game server simulation. Figure 4 presents the adaptation strategy concept using the UML (Unified Modelling Language) notation.

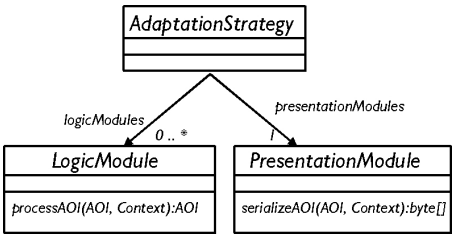


Figure 4: Class Diagram - Adaptation Strategy

According its definition, each *LogicModule* receives the player's *AreaOfInterest* and *Context*. Based on this arguments, the *LogicModule* creates a new *AreaOfInterest* with modified information, using the *processAOI* operation. Examples of logical procedures are:

- **Information Filtering**: based on the device and on the priority of each element of the area of interest, elements might be discharged and not sent to the player. This implies two advantages: *(i)* reduction of network traffic, which might be important for the players using some type of connection which the charge is done based on the amount of bytes tranfered; *(ii)* a smaller processing load and memory demanded from the device, considering that less objects will be processed by the client application;
- **Bi-dimensional Scale**: it adjusts the objects' positions in an area of interest (bi-dimensional) based on the resolution of a target device. By delegating this this to the service, the processing load in the device is decreased;
- **Tiling**: it adapts the objects' positions of an area of interest (bi-dimensional) in order to be presented like tiles-based

games. In these games, a map is composed by bi-dimensional blocks with same sizes that fill it completely and without overlapping. Therefore, the position of any object showed in the map must incorporate a limited amount of values that can be mapped through a template with N lines and N columns;
- **Tri-dimensional projection**: it converts a tri-dimensional projection of the objects' positions into bi-dimensional values. This is very useful in 3D games that need to be presented in mobile devices, which are not suitable to represent tri-dimensionals scenes due to limitations like screen size or processing capacity.

Each adaptation strategy may use as many procedures as necessary. If an adaptation strategy contains more than one *LogicModule*, these procedures are pipelined. Although only four were cited here, the service is extensible and other *LogicModules* can be included.

With regards to presentation procedures, each *PresentationModule* also receives the player's *AreaOfInterest* and *Context*. According to these informations, the presentation procedure transforms the *AreaOfInterest* in a sequence of bytes, using the *serializeAOI* method. Two presentation possibilities were defined:

- **Textual-based**: based on an idiom choosed by the player, this module creates a text-based summary of its area of interest. The idiom is stored as a device property that is acquired from the player's context. The game designer may choose how each textual representation for any game element is defined through the PM2G Game Simulation API;
- **Default-based**: all information about an area of interest is sent in a standardized format, according to the protocol defined by the game designer. This presentation format suggests how the game state is transmitted to the players using desktop PCs.

Each presentation module defines the only format which the area of interest is serialized to each player. Thus, it makes sense that each adaptation strategy contains only one *PresentationModule*.

4.2 Interaction Adaptation Service

As already mentioned, this service follows the idead proposed by Fox [Fox 2003]. Mobile players actions are done inside the virtual world through high-level commands that are executed by the game simulation. This approach means that a mobile player does not have to effectively control his avatar during the whole action. After a command is requested, its avatars behaviour is similar to a NPC. Therefore, mobile player actions might be done without needing a permanent connection or immediate feedback to its commands. This allows the player to trigger an action in the virtual world while he does other activities, such as getting around or receiving calls from his mobile phone. In order to enable this functionality, Figure 5 presents how each action is represented by the Interaction Adaptation Service, using the UML notation.

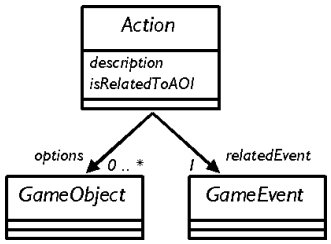


Figure 5: Interaction action - Class Diagram

The service retrieves all possible actions that one player may perform at a given moment from the game simulation. Two factors are relevant for obtaining the possible actions for each player: *(i)* his current avatar state and *(ii)* his area of interest. With regards to the current state, each possible action is seen as a transition between states. For instance, if the player is in a "resting" state, he could change to the "attacking" state or "moving" state. However, certain actions just make sense according player's area of interest. For instance, an attack action would make sense only if there were at least one enemy in the proximities of the player.

Due to this characteristic, actions might also have options. These *options* are alternatives for perfoming an action. For instance, the "attack" action should be associated to all possible targets. In this way, the choice of an action demands also a choice of one associated option. Action options are represent by multiple *GameObject*s associated with an *Action*.

Each *Action* has also an *GameEvent* associated with it. This event reflects the standardized way in which the simulation framework receives actions from all players. Thus, when the Interaction Adaptation Service receives the action chosen by the player, it transforms this *Action* in its associated *GameEvent* and sends it to be performed by the simulation framework.

5 Case Study

The validation of the PM2G architecture was based on prototyping the specified services and a game derived from PM2G scenarios. All PM2G components where implemented using the Java platform in both J2SE and J2ME profiles.

Our experiment used three different platforms: *(i)* deesktop computers; *(ii)* graphically-enabled mobile devices, such as PDAs and advanced mobile phones; *(iii)* textual-based mobile devices, such as some simple mobile phones.

The implemented game is a version of an existing game called Free Wizards[Cecin et al. 2004], adapted to a RPG version. In this new version, each player controls a wizard that lives in world populated by other wizards and mythical animals. Its main objective is to compete through combats and pursuit of objects, such as weapons and treasures. This virtual world is divided into fixed rectangular regions that represent the player's area of interest. Combats between wizards occur through collisions among them. In the same way, wizards can collect objects when they collide with them. This game was renamed Pervasive Wizards and four scenarios were tested with it:

- **Obtaining the game state**: it sends to every player, the contents of his area of interest. The regularity and detailing level depends on player's context. Players using PCs are constantly updated at a rate of 100ms. Mobile players, on the other hand, must retrieve the game state according to their desire. The amount of information varies according to each player's context. Players using desktop PCs receive all the possible information either about the players's avatars, NPCs or virtual items. Mobile players, however, do not receive about the NPCs. The detailing level for each mobile context is different too: players using graphically-enabled mobile devices have an idea of his avatar position inside their area of interest, while players using textual-based mobile devices do not have this information. Figure 6 shows the views that one player could have from the same area of interest;

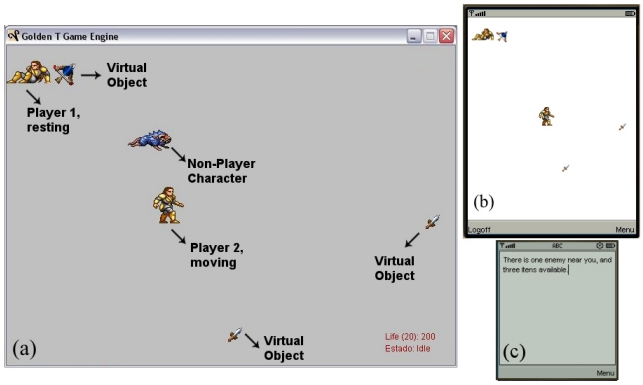


Figure 6: Different views of the game state

- **Avatar's movement**: it allows to move the player's avatar freely across his area of interest or any other part inside the

game world. This can be used in the achievement of a specific objective such as an attack to an enemy. "Moving an avatar" was defined as a low level interaction because it demands that the player controls his character's current position effectively. Therefore, it was implemented only for desktop computers. The player interacts through his mouse, clicking on a new virtual world position for his avatar, which will move towards this new position;

- **Attacking an Enemy**: this is an example of interaction between two players in which their avatars fight against each other. For PC players, there is not an explicit command for this action. Rather, the player has to control his character and guide it towards the opponent. Meanwhile, the mobile player, cannot control directly his avatar movements. But, he may choose this command from their available list of actions. His avatar will move in an automatic way towards the opponent and they will start to fight against it.

- **Collecting an item**: this happens when a PC player moves his character towards the aimed item. This is similar to an attack against an opponent. A mobile player has an explicit command when there are virtual items available in his area of interest. When the action in sent to the server, the game simulation moves the character up to the virtual item to cause a collision and therefore, the item collection. Figure 7 shows a mobile player's view during an action of collection.

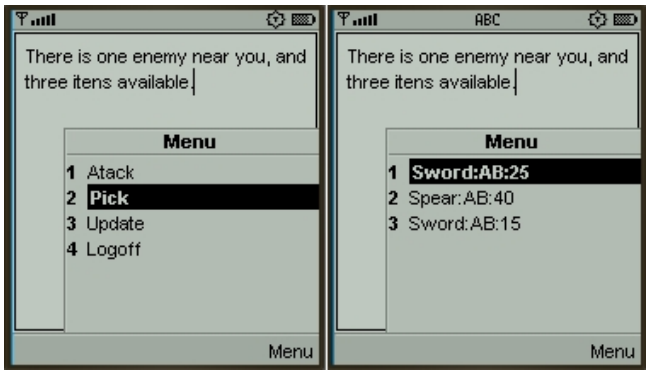


Figure 7: Choice of actions via a mobile device

5.1 Services Configuration

In order to run our experiments, both content and interaction adaptation services were configured in accordance with PM2G scenarios. Three adaptation strategies were defined. Desktop PCs' clients do not realizes any logical processing over the area of interest and uses the standard presentation procedure. In other words, all information about the player's area of interest is sent using a custom protocol defined by the game.

Graphically-enabled mobile devices also use the standard presentation procedure, however its area of interest is previously changed by two logical procedures: bi-dimensional scale and information filtering. In other words, the area of interest is re-dimensioned according to the device dimensions and irrelevant information about NPCs are discarded. Finally, the adaptation strategy for text-based mobile phones uses the same logical procedures described earlier, but uses a textual presentation procedure.

Regarding to Interaction Adaptation Service, the game must supply two configuration arguments: *(i)* an association among states concerned with different platforms and *(ii)* a mapping from existing states to possible actions for one player. Pervasive Wizards defined seven states. Four states were defined for PC players: Idle, Moving, Fighting and Resting. Three states were created for mobile players: MobileResting, MobileFighting and MobileCollectiong. According to his current avatar state and his current context, the player receives all possible actions and options (if available) that he could do.

6 Evaluation

The implementation of both PM2G architecture and the Pervasive Wizards game enabled us to perform some experiments. First, we collected performance data about PM2G main services. We also collected information about the amount of data exchanged among services and clients. Finally, we aimed to discover how much the PM2G API helps the developer in a crossmedia implementation. Next subsections shows our results.

6.1 Performance

In order to evaluate PM2G middleware, we logged the time elapsed for both content and interaction procedures in PM2G services. Our experiment was conducted using two desktop PCs, each one with 1GB RAM and Pentium 4 processors. One PC hosted both the Game Server and the PM2G services; the second PC was used to simulate multiple players. We used J2ME emulators to simulate mobile devices. The PC client application was changed to perform an automatic behaviour, where its avatar moves to a random position every five seconds. Simulated avatars may fight or collect virtual itens while moving randomly. Since we had interest in these situations, the virtual world was compounded by four (2x2) regions only. Figure 8 illustrate how the experiment was conducted.

Everytime a player requires an update about his area of interest, his client application retrieves this information from the Game Server (step 1). The Game Server uses both Content Adaptation (step 2) and Interaction Adaptation Service (step 3) to determine how information about the game state must be sent to each player. Both services know the current context of each player because they are subscribers of the Context Managment Service. So, everytime a player changes his context (step 4), both services are notified (step 5). This avoids that both services inquiries the context managment service constantly. The game server receive the serialized data from both services, assembly these information into an unique update package and send it back to the players client application(step 6).

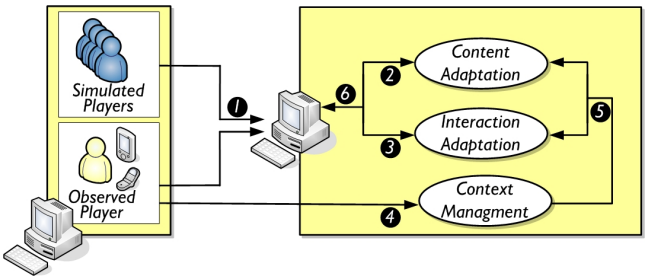


Figure 8: Choice of actions via a mobile device

During our tests, it was generated a log file containing data about an specific player, which used all possible devices while playing Pervasive Wizards. This data includes: *(i)* Time elapsed for content adaptation procedures; *(ii)* Time elapsed for interaction adaptation procedures;*(iii)* amount of data exchanged among services and game clients. In this experiment, 4740 samples were collected during five minutes of game session. We simulated up to 10 players, and both PCs remained with their CPU load below 90%.

6.1.1 Execution Time

The first analysis was made based only on the time spent in adaptation procedures inside both services. Figure 9 show the time spent with content adaptation procedures for the observed player, while he used different platforms. According to our results, the average content adaptation time when the player was using a PC was 51300.66ns (Standard Deviation = 15214.24ns). For mobile devices, this average time increased to 103231.85ns (Standard Deviation = 21368,52ns). This behaviour reflects the fact that mobile players needed extras logic procedures for scaling and filtering their area of interest.

Concerning the interaction adaptation service, our experiment showed that the service spent 13648.38ns (Standard Deviation =

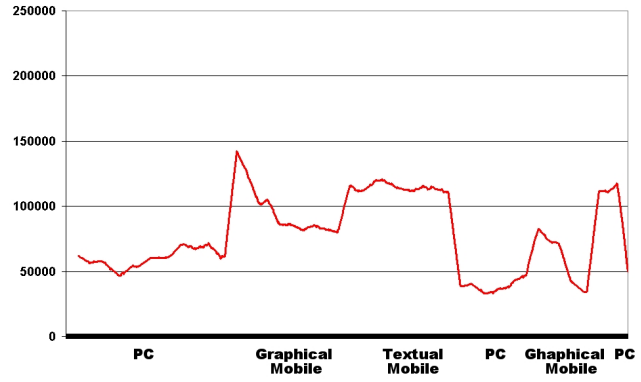


Figure 9: Time Spent for Content Adaptation Procedures

7564.10ns) on average, when the player was playing using a PC. Figure 10 presents our results. Using a mobile device, this time increased to 64374.53ns (Standard Deviation = 9142.69ns). This difference reflects that no actions needed to be retrieved when a player was using a PC. For both textual and graphical mobile devices, the procedures were the same, and time elapsed was concerned with the amount of information insied the players area of interest.

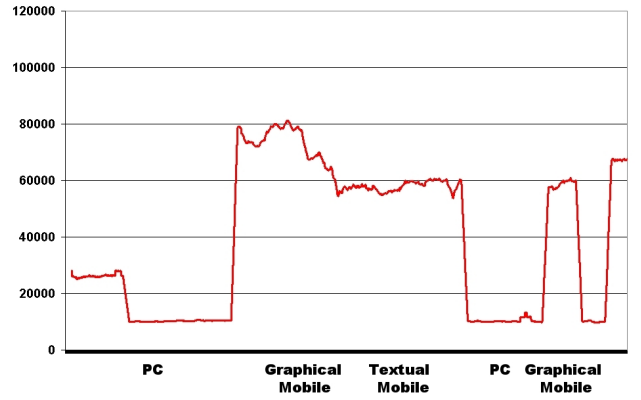


Figure 10: Time Spent for Interaction Adaptation Procedures

6.1.2 Exchange of Data

The last analysis was done regarding the amount of information exchanged among services. With regard to Content Adaptation Service, the number of elements inside the players area of interest is the determinant factor either for desktop PCs and graphical mobile devices. The reason for that is that the standard presentation procedure associated with their adaptation strategy. For textual mobile devices, all information about players area of interest is transformed in a fixed-length text message, regardless the amount of objects inside their area of interest. Figure 11 shows the amount of bytes transmitted by Content Adaptation Service, for the observed player, while he played with different devices.

With regard to Interaction Adaptation Service, no information about actions needed to be sent when the player was using a PC. However, when using a mobile device, the amount of information is directly dependent on the number and type of elements inside the player's area of interest. These results revealed that the need of information about the avatar's interaction creates an unwanted increase of the data transmitted to mobile players. Figure 11 shows results that prove this analysis.

6.2 Qualitative Evaluation

Our experiment also allowed us to evaluate how much the PM2G middleware helps developers to implement RPG crosmedia games. In order to do this, we classified each class used in our case study

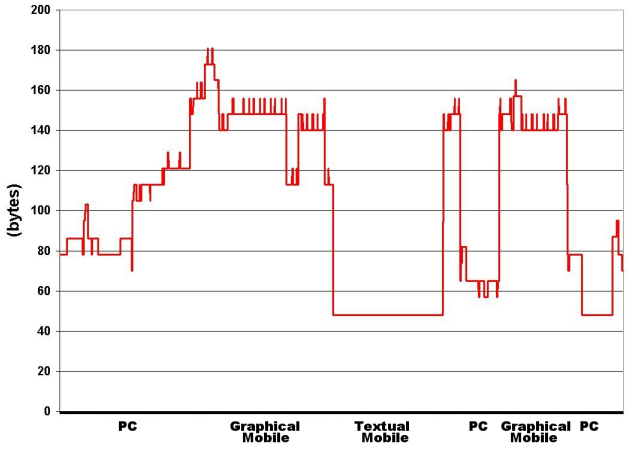


Figure 11: Exchange of data for Content Adaptation Procedures

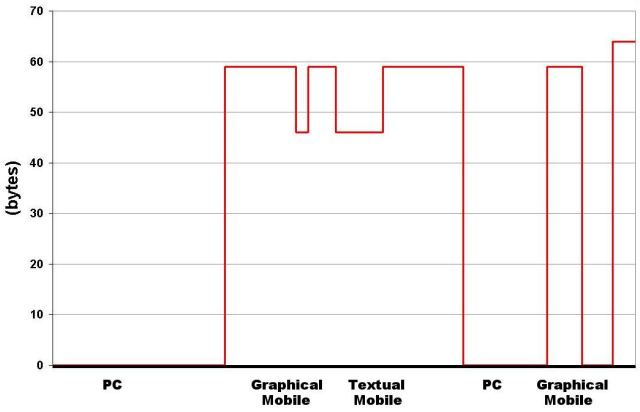


Figure 12: Exchange of data for Interaction Adaptation Procedures

in two categories: game-specific classes and PM2G generic classes. The former contains all classes especially created to implement the Pervasive Wizards Game. The latter includes classes that can be reused in other games. Results are presented in Table 1.

According to our case study, 207 classes were needed in our experiment. Our results highlights that more than 150 classes were generic, which represents 75% of all effort demanded to build our solution (See Figure 13). These classes will not demand changes when new PM2Gs needed to be constructed. Most of Pervasive Wizards specific classes were concerned with the simulation framework (classes used to represent wizards, NPCs, objects and states). We also used other metric to evaluate the effort of using the PM2G middleware: the amount of source lines of code (excluding blank lines and comments). According to our results, presented in Figure 13, almost 60% of the code used in our experiment is from de PM2G middleware. Comparing to results obtained from the first metric (number of classes), we conclude that Pervasive Wizards specific classes are more coarse grained than the generic classes.

7 Conclusions

Crossmedia games represent a view for future digital games. These games can be seen as a pervasive application, where players would be able to enjoy the game from almost anywhere, at anytime, using heterogeneous devices.

In this document, we present the PM2G architecture, a support platform for crossmedia games. This service-based architecture includes five services that aims to attenuate the development and execution of RPG crossmedia games. Particulary, we evaluated two services: Content and Interaction Adaptation Services. These services guide players participation, according to heterogeneous devices.

Table 1: Middleware Evaluation

	Pervasive Wizards		PM2G Middleware	
	#Classes	%	#Classes	%
Utilities	1	1.96	0	0.00
Communication Framework	11	21.57	44	28.21
Simulation Framework	24	47.06	21	13.46
Context Managment	0	0.00	21	13.46
Content Adaptation	1	1.96	12	7.69
Interaction Adaptation	5	9.80	42	7.69
Messages Notification	6	11.76	48	11.54
Mini-Worlds Integration	3	5.88	28	17.95
Total	51	100	156	100

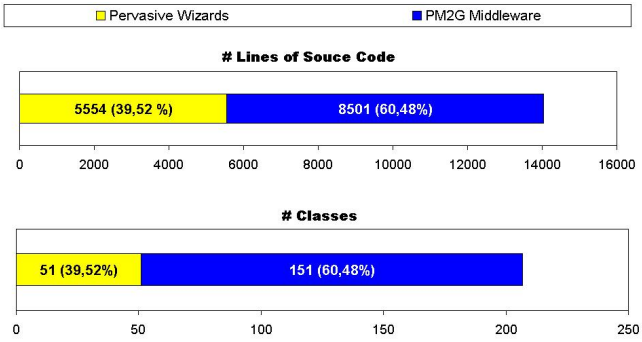


Figure 13: Metrics for PM2G Evaluation

Our case study proves that both services are efficient based on a benchmark analysis of time elapsed to perform their funcionalities and the amount of exchanged data among players and the game server. An analysis about the its API show that the PM2G solution has a significant impact and may reduce the development effort for crossmedia games based in PM2G scenarios.

In our future studies we intend to offer the PM2G middleware to game designers and developers evaluation. By doing so, we expect to collect their feedback about the API quality based on its easiness of use.

References

CECIN, F. R., DE OLIVEIRA JANNONE, R., GEYER, C. F. R., MARTINS, M. G., AND BARBOSA, J. L. V. 2004. Freemmg: a hybrid peer-to-peer and client-server model for massively multiplayer games. In *NetGames '04: Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games*, ACM Press, New York, NY, USA, 172–172.

FOX, D. 2003. *Massively Multiplayer Game Development*, vol. 1. Charles River Media, Inc., Rockland, MA, USA, ch. Small Portals: Tapping into MMP Worlds via Wireless Devices, 244 – 254.

HAN, J., IN, H. P., AND WOO, J.-S. 2004. Towards Situation-Aware Cross-Platform Ubi-Game Development. In *APSEC*, 734–735.

KOIVISTO, E. M., AND WENNINGER, C. 2005. Enhancing player experience in MMORPGs with mobile features. In *2nd International Digital Games Research Association Conference*.

LINDLEY, C. A., 2005. Game space design foundations for trans-reality games: Keynote paper. Advances in Computer Entertainment (ACE): Special Track on Game Development, Barcelona, Spain, June 2005.

LINDT, I., OHLENBURG, J., PANKOKE-BABATZ, U., AND GHELLAL, S. 2007. A report on the crossmedia game epidemic menace. *Computers in Entertainment 5*, 1, 8.

MENEZES, A., SILVA, C. E., ARRAES, D., PEDROSA, D., BRAYNER, F., TRINTA, F., WANDERLEY, I., MACHADO, M., BORGES, R., AND RAMALHO, G. 2006. Uma plataforma para jogos mveis massivamente multiusurio. In *V SBGames, Brazilian Symposium on Computer Games and Digital Entertainment*, Federal University of Pernambuco.

PEDROSA, D., TRINTA, F., FERRAZ, C., AND RAMALHO, G. 2006. Servicos de adaptacao de jogabilidade para jogos multiplataforma multiusuario (in portuguese). In *V SBGames, Brazilian Symposium on Computer Games and Digital Entertainment*, Federal University of Pernambuco.

TRINTA, F., FERRAZ, C., AND RAMALHO, G. 2006. Middleware services for pervasive multiplatform networked games. In *NetGames '06: Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games*, ACM Press, Singapore, 39.

TRINTA, F., FERRAZ, C., AND RAMALHO, G. 2006. Uma proposta de cenarios e servicos de suporte para jogos multiusuario multiplataforma pervasivos (in portuguese). In *WebMedia '06: Proceedings of the 12th Brazilian symposium on Multimedia and the web*, ACM Press, Natal, Rio Grande do Norte, Brazil, 243–252.

TRINTA, F., PEDROSA, D., FERRAZ, C., AND RAMALHO, G. 2007. Scenarios and middleware services for pervasive multiplatform networked games. *PerComW'07 - Fifth Annual IEEE International Conference on Pervasive Computing and Communications Workshops 0*, 593–596.

WOODCOCK, B. S., 2006. An Analysis of MMOG Subscription Growth. Available at http://www.mmogchart.com/.

moBIO Threat: A mobile game based on the integration of wireless technologies

Wilian Segatto Eduardo Herzer Cristiano L. Mazzotti João R. Bittencourt Jorge Barbosa

Universidade do Vale do Rio dos Sinos (UNISINOS), São Leopoldo, Brazil

Abstract

Pervasive gaming is a new genre of gaming that became possible because of the development of the communication technologies, especially wireless ones. In this area of gaming, players must physically walk to certain places of the game area to reach their objectives and missions. They may also interact with the environment and with real objects. Nowadays, there are just a few pervasive games developed, and all of them have limitations concerning localization tracking, hardware flexibility, signal coverage and cheap set-up. In this context, an innovative game, moBIO Threat // Disease Control, was developed, based on the integration of multiple wireless technologies, mixing their capabilities and neutralizing their limitations. It utilizes RFID, IrDA and QR Code technologies for object interaction, Bluetooth for exchange of information between players physically close and the IEEE 802.11 Wi-Fi protocol to connect all of the players with the game server. From what could be seen on the played matches, moBIO Threat brings a completely different game experience; social gaming and collaboration are a strong attribute of the game. This paper describes the game development other details about the project and about pervasive gaming.

Keywords: pervasive, social gaming, Bluetooth, RFID, QR Code

1. Introduction

Pervasive gaming is an emerging genre of gaming, in which players must physically move to specific places in order to perform tasks within the game. Users must interact with each other and with the environment around them. To accomplish such interactions, communication technologies must be used in order to allow users to send and receive environmental information such as location and users nearby, for example. Players may interact with real objects – whether they are simple handheld objects or large rooms. Usually, the technologies used on these type of games are wireless, because they allow players to freely walk, without having to worry about plugs and wires. Usually, the most commonly used protocols are IEEE 802.11, Bluetooth and GPRS.

There are a few pervasive games already developed, such as *CatchBob!* [Girardin 2005], *Uncle Roy All Around You* [Benford et al. 2003], *Human PacMan* [Cheok et al. 2003] and, most recently, *Plundr* [Plundr 2007], each of one with their own private peculiarities and features. The common characteristic among them is that they were created focused on one – and only one – interaction technology.

The main goal of the *moBIO Threat Project* is to create, by the use of a conversion of communication technologies and the integration of their specific capabilities, an innovative game with improved gaming experience on both interaction and localization tracking domains. Besides, the game was meant to be educative. In other words, its objective is to teach users about one or more subjects while they were playing.

The game is also meant to be distributed, so users can set it up in their homes and schools. Furthermore, it was taken in consideration that most players probably would not have available all of the technologies utilized on the project. Based on this, an adaptation layer was implemented to provide minimum loss of functionality in case of the absence or failure of one or more technologies.

This paper describes the development of the moBIO Threat game, a pervasive game based on the use of multiple technologies. The paper is organized in five sections. Section two describes the game plot and some features of the game, as well as the game modeling. Section three presents the game development and implementation. Related works are described on section four, and conclusion on section five.

2. moBIO Threat

The first stage consisted in looking for communication technologies that could provide exchange of data between players with themselves and with the environment. Since players had to walk, wire-based means of communications were completely discarded since the beginning of the research.

The second stage consisted in creating the game plot, taking in consideration all of the technological and educational aspects specified on the project proposal.

The game consists of two competitive teams trying to accomplish their own missions while trying to prevent the other one from achieving its goals. The

genre of the game is pervasive; players have to physically walk in order to perform the tasks required in the game.

The chosen name, *moBIO Threat*, combines two words: “mobile” and “bio” (from “biological”). The pronunciation of “moBIO” is the same as the British for “mobile”; being “bile” read as “bio”, from “biological”. So, the name “moBIO Threat” means both “mobile threat” and “biological threat”.

2.1. Game plot

The game takes place in a university devastated by a major terrorist attack that destroyed most of the facilities and communication infrastructures between the campus and the rest of the world. Since the ways out have been blocked, the terrorists who remained in the area and the military unit that was giving a speech got trapped in a confined area of the university. The terrorists (TRs) attacked and kidnapped the biologists of the university's microbiology lab and also stole their research from the past ten years. The research was about the development and the use of pathological agents in rats. Therefore, the terrorist now have the potential to build a deadly toxin and disseminate it. The counter-terrorists’ (CT) mission is to prevent the enemy from achieving a mutated pathological agent from the original one and launching it using a missile. This can be done by either neutralizing the enemy force or by synthesizing an antidote for the agent being developed.

The players start the game round in a waiting room in which the teams are defined. Next, the teams must report to their bases, so they can receive their first assignments.

The missions are the mean by which teams can earn money and equipments to help completing their main goals in the game. The missions are different for each team. TRs’ missions include choosing the pathological agent to be developed, searching for its requirements – such as chemical substances – and building a laboratory to synthesize it. For CTs, the missions include finding the kidnapped biologists and gathering information to discover what agent is being developed by the Terrorist team, permitting the development of an antidote.

2.2. Game features

There are several items that can be bought during the game to help the teams to achieve its goals during the game. To do that, players must request their commander to acquire the wanted items. After achieving them, the commander must give the demanded items to the player working as the supplier, so he or she can take them to some place or distribute them to the other players. Basically, there are two kinds of items to be bought. The first is weapons; users can upgrade their armory, obtaining new and improved weapons to help defeating the enemy while in combat. The weapons are pistols, rifles, shotguns, sub-machineguns and explosives charges such as C4. There also the missile components, from its fuel and engine to its body and warhead.

During the game, there are some places that can help the teams to achieve its goals. The first and most important location is the base, where the team commander controls the other players and where players can recover themselves from battle injuries.

The other locations can be constructed while the match takes place. The Supply Center is like a small storage, where armory and other items can be stored and retrieved afterwards. The Silo is the place used for assembling the missile and launching it, while the Bunker is a protected location against the missile and its pathological agent. The Laboratory is used to synthesize the agent and the Intelligence Center, which can be constructed only by CTs, retrieves information of the other team.

When two or more enemy players are physically close to each other, they can initiate a combat, utilizing their personal armory. If the loser does not die and chooses to surrender, he or she is taken to the enemy prison. If he/she dies, he/she gets out of the match.

2.3. Game model

The game structure is divided in layers and modules. Figure 1 shows the game modularization diagram.

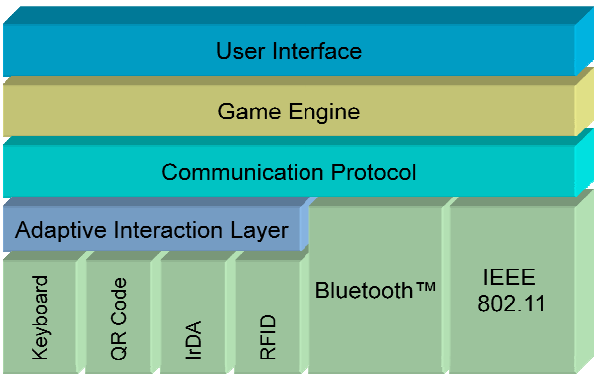


Figure 1- Game modularization diagram

The User Interface is the most high-level component of the game. It is responsible for rendering the game elements such as maps and user and team information. It also handles the user commands and activates the corresponding module on the Game Engine. The User Interface also handles alerts and messages to and from the players.

The Game Engine is the core of the software. It is connected to the User Interface and to the Communication Protocol and it is responsible for the processing of the game, for controlling and interpreting the information that is acquired from the game server and for updating the game interface.

The Communication Protocol handles the exchange of information between the players with each other and with the game server. It is explained in details on the section 3.2.

The Adaptive Interaction Layer is the mean by which the communication technologies utilized to interact with the world are centralized and made transparent to the Communication Protocol. This module is explained in details on section 3.8.

The remainder modules are technology specific and refer to as the hardware-software integration and are described on section 3.

3. Implementation

The game can be played by up to eight players. Each team has a Commandant, responsible for assigning the missions to the other players and for controlling the achievement of items. Other players have functions within the team; they can work as Supplier, Medic or Engineer.

The Supplier is the responsible for requesting the acquirement of armory and items, getting them from the Commandant and taking them to specific places or players. The Medic is the player that can heal his/her team players and himself/herself. The Engineer is capable of building the support locations for the team.

3.1. Personal equipment

Each player carries his or her personal device, a Tablet TC1100 from HP. The Tablet has a Celeron M 1.0GHz processor and 256MB of RAM, with built-in loudspeaker and microphone. It also has built-in Bluetooth and Wi-Fi (IEEE 802.11) cards. Players can use a removable keyboard to type their names and login information when the match is about to start, but it is detached afterwards, to minimize the weight of the mobile equipment. Because of the absence of a physical keyboard and mouse and to make easier the inputs, players utilize the pen of the Tablets to interact with the game software. Figure 5, at section 3.8, shows the game running in the device.

3.2. Game software

The game software was developed using the Java language, conforming the J2SE version 5.0 libraries and specifications. The game interface – as it can be seen in Figure 2 – consists of four main sections. The left column (Number 1 on the figure 2) shows the *player rank*, the items being carried in the *personal bag* and the *wanted items* for the current mission. The lower left corner (Number 2) has information about the player, such as *name*, *team* and *health*. The *health* level is presented in a graphical percentage-based bar. The bottom of the screen (Number 3) has information about the events and warnings of the game and gives feedback about the status of the connected interaction devices, such as RFID and Bluetooth. The section also features a search-for-enemies button, which enables the scan for enemies to permit a combat between the players. The bigger section (Number 4) contains the map of the game area, which shows information about the location of friends, places and items in general. The localization tracking method used on the project is described on section 3.9. The localization of the game components on the map is updated when it is detected the change of their positions, utilizing a coordinate-based module.

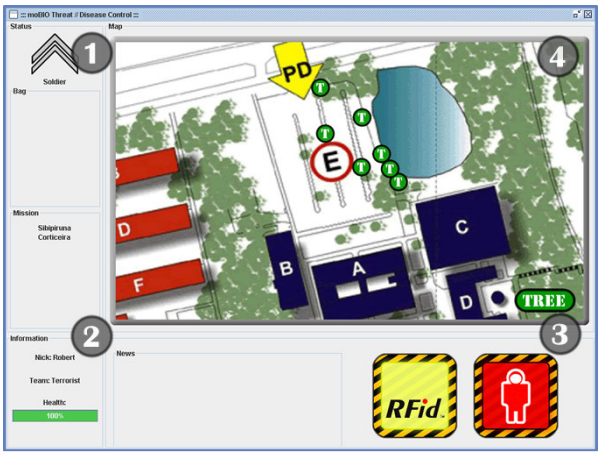


Figure 2- Game main screen

Since the beginning of the development of the game, the communication protocol was meant to be as well optimized as possible. The exchange of data between the players and the server and between the players with each other is based on keyword bytes and the content of the messages. There are around 15 keywords, each one representing one type of object being transferred on the network; and there are two kinds of communication: client requests and client dispatches. The firsts refers to the situations where the client needs information or an operation from the server such as requesting the current mission of the team and login/logout operations. The other happens when the client wants to send updated information about him/herself to the server.

The exchange of data initiates when the client-side of the connection sends to the server one of those keywords. If it is a client request, it waits until the server replies to the request; if not, the client also sends the content of the dispatch such as text, numeric values or more complex data such as specific objects.

On the server-side of the connection, the game server features a multi-task listener that handles all of the requests and dispatches; it is always listening for a keyword byte to arrive and behaves differently for each one. Whenever a known code is received, the server executes the operations of the task required: if it is a client request, it sends the requested information; if not, it performs another reading, to acquire the content of the message.

Because of its optimization, the communication protocol coding is more complex and it was more difficult to be developed. On the other hand, the network bandwidth required is insignificant.

The moBIO Threat server also features two important modules, the Event Server and the Coordinate Manager.

The first module informs the players about the game status, it is responsible for routing messages and alerts to the players. These notifications are stored on the match log and are sent to specific players, depending on its origin. "Game started", "Item acquired" and "Player arrested" are examples of the messages that can be sent. The notifications are only for information purposes, events of position change are handled by the Coordinate Manager.

The Coordinate Manager is the module responsible for updating the clients' maps based on localization events. The information required is taken from user interaction events such as entering a room or getting an item. The Localization tracking protocol is specified on section 3.9. Since the game is meant to be set up and played in different places, it is necessary item and room set-ups on the first time the game is played.

The team commandant also has an administration module, from where he or she can send orders to the other players.

3.3. Wireless

The communication between the players and the server are established by a Wi-Fi IEEE 802.11 Network. The Access Point utilized was a Cisco AIRONET 1100. To provide maximum range and signal coverage, the AP was placed in the corner of the 'B' building of the Center 6 of the University, as pointed in the Figure 3.

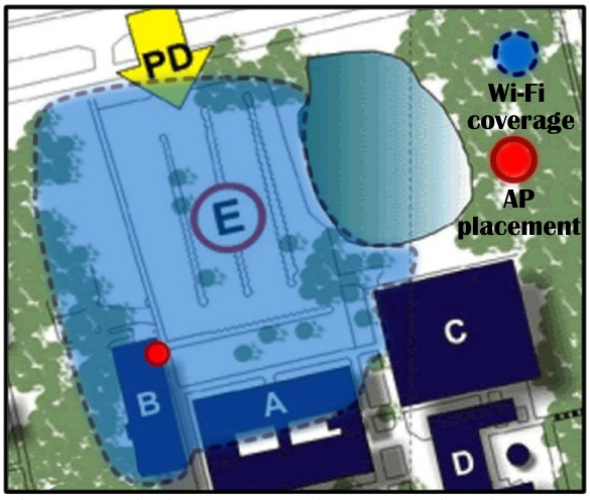


Figure 3- Wi-Fi coverage and Access Point placement

The Access Point provides maximum range of 100 meters and covered an area of around 7000 square kilometers.

The Wi-Fi technology is the main mean of communication utilized on the game, because the users are within the range of coverage almost the whole time. Events of localization and messaging are communicated through the Wi-Fi connection.

Because of the trees and the walls on the buildings, players may, eventually, enter a dead spot, and cause the connection to terminate. However, a reconnection module was developed to guarantee the process of reconnection as soon as the user leaves the dead spot. When this loss of signal happens, the map becomes unavailable and a static signal animation takes place, informing the player about his/her loss of signal.

3.4. Bluetooth

The Bluetooth technology is utilized in the project to provide interaction between the players when they become physically close to each other. The *avetana* [AvetanaBluetooth 2007] library was used to integrate the Java software and the built-in Bluetooth card. The utilized protocol was the RFCOMM.

Usually, the Bluetooth scan engine is very slow, because it has to find all devices in range and then look for active services on them. Initial tests reached results of up to 20 seconds to find an enemy. This amount of time was too large, causing loss of the game dynamism. Further optimization and tests reduced the search and connection times by the half, 10 seconds. This amount of time can be reduced even more if a more complex search structure were implemented.

If a player wants to battle, he or she must click the "Enemy Search" button (Figure 2, bottom right corner) to initiate the search and engage a battle. The player that looses the battle may leave the match or may be taken to the prison of the enemy team.

If a player is utilizing a device that does not have Bluetooth technology available, despite not being able to battle enemies, he or she can still play the game and interact with objects. The communication with the server and with other players – established through IEEE 802.11 (Wi-Fi) technology – is not affected or compromised at all.

3.5. RFID

The Radio Frequency Identification (RFID) technology is another technology present on moBIO Threat. The RFID tags are meant to be stuck in the game objects such as trees and rooms. To read the tags, players must be carrying a RFID reader connected via USB connection to the Tablet, and approach it to the tag. The RFID reader used was the Texas S4100 MFR (Multi-Function Reader) Evaluation kit and the tags were plastic ones, as shown on the Figure 4. The tags are passive, so the distance with the reader has to be of some centimeters.

Figure 4- RFID tag on a tree

3.6. QR Code

QR Code is a technology that graphically stores data in a 2-dimensional matrix. In some square centimeters, a QR Code tag can store up to 7KB of data. Although it demands a camera to be present to accomplish the tag reading, tags can be printed using any kind of paper and a regular printer (such as an inkjet one). QR Code is utilized to provide object interaction.

3.7. IrDA

The infrared technology is also supported by the game. Since the Tablets have built-in IrDA ports, no additional hardware is necessary. The module to be placed on the objects, however, is more complex and demands specific hardware and microcontroller programming and a battery.

3.8. Adaptation Layer for Object Interaction

This is the module that communicates with the technology-specific hardware-software interfaces. Its function is to permit object interaction even when one or more technologies are not operating or not present. Object Interaction will work with the available technologies for the user. For instance, if the user does not have a camera for QR Code, moBIO Threat will automatically detect its absence and disable the QR Code module.

It could happen that the player does not have any of the specified technologies. When this happens, he or she can still interact with the objects, typing with a virtual keyboard the alphanumeric code provided on the item modules.

The item module that supports all of the technologies is still under development, but its final form will be a box-shaped hardware component with: a QR Code tag on the outside; a RFID tag inside it with a indicative marker of its position on the outside; and a embedded IrDA module and battery with the infrared LED partially outside and a indicative marker of the LED position. The alphanumeric code for typing is printed together with the QR Code tag.

When an item is detected, its hardware components activate the listeners on the game software. The identification numbers are obtained and the Adaptation Layer is activated, generating an output message that is based on a common standard. The event message is transmitted to the server, that queries the database for the item that corresponds to the unique code sent. The item is then dispatched to the client, and a confirmation screen appears, as can be seen on Figure 5, where it is being presented to the player the message “Do you want to enter room 202?”

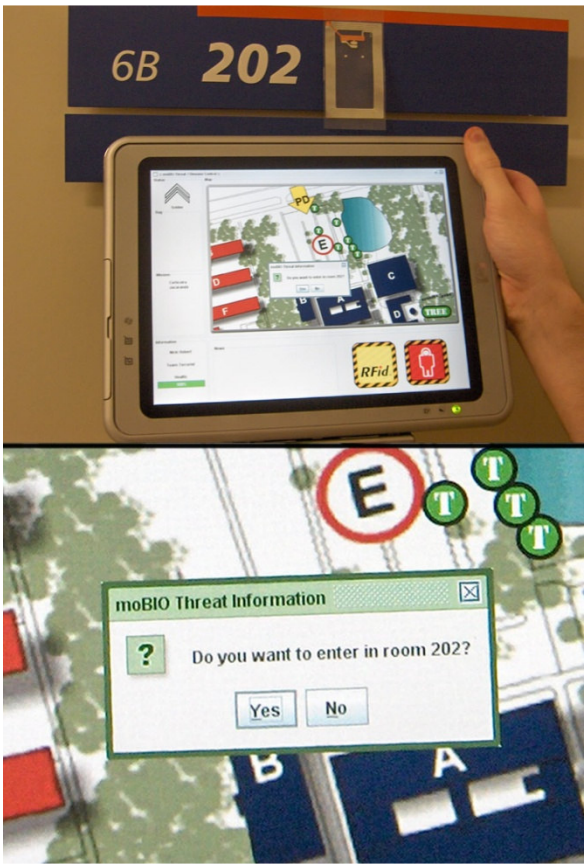


Figure 5- Interaction with rooms. “Do you want to enter room 202?” message being displayed.

3.9. Location tracking

This is a very important aspect of the pervasive gaming area. There are a lot of tools that can help improving the accuracy of the tracking, such as multiple Wi-Fi antennas and ultra-sound or radio-frequency sensors, to name some. The problem with this kind of tracking is that they demand complex installations and location training and they need, on most of the times, specific and relatively expensive hardware to be installed. The GPS system was also discarded because it does not cover indoor locations – a very desirable situation to the game.

Since the project was designed to be portable, the methods of localization tracking were simplified; the

change of position only is detected when the user interacts with objects and rooms, which are already mapped and registered.

Future works will give the players ability to set their current localization by clicking on the map. This way, players will be able to inform their location to the team players, facilitating the request for support or trade of items, for example.

Concerning dynamic maps – that can be drawn depending on where the game takes place, a module will be developed to permit easy registration and localization of objects and rooms.

3.10. Game experience

The match starts with all of the players in the Start Room, where the teams are defined and then go to different start locations. At this point, the server sends the *game start* signal to all of the players, and teams receive their first assignments. Then, the search for items or places begins, and the players start to explore the game area to find them. As they interact with objects, their position on the game map is being updated, so players can know where the team players are. The items can be taken to specific rooms that work like the team laboratory, for example – this way, missions are being accomplished. During the match, players can activate the *enemy search* button, which triggers a scan for players nearby and makes the combat module to be activated if at least one of them is found.

moBIO Threat features a way of playing completely different from what people are used to. Many times, during the game, players must look for specific objects or places. They have to walk and explore the game area to gather the wanted items, which are always real objects; users can see and touch the game elements such as trees and rooms.

Since the team players have the same objectives, they are constantly talking to each other, setting up strategies and trading information about items and enemies. For this reason, there is a strong sense of collaboration between team players. There is also an enhanced competition sense, since enemies can be seen, followed and battled.

The interesting is that the game provides, automatically, virtual feedback for actions being taken on the real world, such as walking or interacting with objects. The map can be used to find those objects and also to locate team players, following and encountering them.

moBIO Threat expectations were accomplished. The game brings an interesting, captivating and innovative way of playing. Figure 6 shows players at a moBIO Threat match.

Figure 6- Players at a moBIO Threat match

4. Related Work

BlueGame [Solon et al. 2005] is a game that features a 2-dimensional multiplayer combat based on the Bluetooth technology. It's played by two players competing with each other. The game consists in choosing three cards and their priority of use; each card has its own power and amount of energy required to inflict the power. The objective is to decrease the opponent's *health points* down to zero. Unlike moBIO Threat, BlueGame was designed to be a game playable only by players close to each other.

CatchBob! [Girardin, 2005] is a collaborative multiplayer game, in which the players participate of a virtual hunt to catch "Bob", a virtual character on the game area; players have to surround it to win the game. This is made through the execution of the strategies set-up on the beginning of the game and by communication by drawings. Players use Tablet PCs and iPaqs to play the game. Players are connected by a Wi-Fi (IEEE 802.11) network, whose antennas and access points also work to provide means of localization. Accuracy and coverage values were not made available. In this game, players just walk to certain places and then surround – or not – Bob; then, they have to go back and restart the match. moBIO Threat features object interaction, which makes the game substantially more attractive by adding the possibility to change the course of the game through objects or places at specific locations.

Uncle Roy All Around You [Benford et al. 2003] took place on the city of London, England. The objective of the game was to find "Uncle Roy", by following the given clues and physically walking around the city. In exchange for their personal possessions – to increase sense of disconnection from the everyday experience of the real word –, *Street Players* were given a handheld computer to begin their search. They could communicate with each other through audio messages. *Online Players*, who were also given hints, could help Street Players. They could communicate with other Online Players by using a webcam. The game ends in a physical office where

players are invited to enter (if they discover it) and then they are taken to a limousine in which they establish a contract to help other players. The player's position were defined by the own players, sliding a "me" icon and clicking an "I am here" button. Besides being an unsecure and inaccurate mean of localization, it was not a problem since the hints were given just in specific locations – and had no meaning if the player was in another place. Networking was established by a GPRS connection. The difference from moBIO Threat is that "Uncle Roy All Around You" could not be played more than once or twice by each player, because the paths to Uncle Roy were always the same. The game does not feature object interaction and localization tracking at all.

Human PacMan [Cheok et al. 2003] is a pervasive version of the arcade game developed in 1980, PacMan [Knight 2004]. It utilizes augmented reality to construct paths by which the player acting as "PacMan" has to walk, colleting virtual cookies on his or her way. Other players, that play the role of "Ghosts", can pursue PacMan and catch him/her. The game uses Bluetooth to interact with objects that can provide the *ghost devouring ability*. Players have to get hold of the objects (usually cans) to enable a capacitive sensor that activates the embedded Bluetooth device inside it. The positioning system is based on the Global Positioning System (GPS) and networking is established through the IEEE 802.11 protocol. Players make use of a light weighted wearable computer that are composed of a head-mounted display (HMD), a video camera, a Wi-Fi card, a hard-drive, a GPS antenna, batteries and other modules. The problem with using GPS is that, despite of having a good accuracy at outdoor locations, it is a technology that has coverage issues i.e. it cannot be used in indoor location or at places with high buildings nearby. As it was said before, moBIO Threat discarded a GPS-based localization system because it was designed to be played in *any* location, indoor or outdoor.

Another pervasive game is Plundr [Plundr 2007], an adventure game where players sail from island to island aboard a pirate ship buying, selling and battling for goods [Kotaku 2007]. However, the available islands to sail are determined by the physical position of the player. Plundr is available for PC and will be supported by Nintendo DS. The localization tracking is based on Wi-Fi Positioning System (WPS). There are no further details about the project, since it is still under development. The point is – as well as moBIO Threat – Plundr can be played in several locations, increasing the number of potential players.

5. Conclusion

Pervasive gaming is a challenging domain, because it deals with networking protocols, game design, game planning and social interaction and collaboration. Network modules must be tough enough to handle events such as undesirable losses of connection, situation where a reconnection protocol must be implemented. The game design has to be intuitive and easy to use; players should not need to read large texts while playing, for instance. In other words, the game interface should not bring any loss to the game dynamism.

The hardware that players carry also has to be as light as possible; the game should not be tiring, it should be interesting and engaging.

However, the technologies that permit pervasive gaming to exist have some imperfections. They are not ready – or simply are not designed – for gaming. For this reason, complex frameworks have to be implemented in order to get better results from the infrastructure that the technologies provide.

A major issue for pervasive gaming is the localization tracking. It is very difficult – and expensive – to cover mixed areas such as cities, fields and indoor locations and to track players and objects with accuracy of up to a few meters. There are a lot of tools that provide certain tracking capabilities, but they are not simple to set up and to configure.

An important aspect of moBIO Threat is that it is an educational game. During the game, the players have to work with pathological agents and chemical reactions for them, for example. This way, players must gather information about game elements by learning how domain-specific subjects work. The areas of Chemistry and Biology are the most utilized on the game educational objectives.

But the most notable aspect of a pervasive game is, undoubtedly, the social interaction and collaboration. As it could be seen in moBIO Threat, being able to talk to other players, to create and exchange game strategies, to actually see what players are doing and where enemies are bring a gaming experience completely different from what people are used to. moBIO Threat provides the sense of physical interaction, the sense of touching real objects that are part of the game.

About the game infrastructure, it was provided a cheap, flexible and easy to set-up game environment. The only technology that is mandatory is IEEE 802.11 Wi-Fi.

References

avetana JSR-82 implementation [online]. Available from: http://www.avetana-gmbh.de/avetana-gmbh/produkte/jsr82.eng.xml [Accessed 15 July 2007].

BENFORD, S., FLINTHANM, M., DROZD, A., ANASTASI, R., ROWLAND, D., TANDAVANITJ, N., ADAMS, M., ROW-FARR, J., OLDROYD, A. AND SUTTON, J., 2003. *Uncle Roy All Around You: Implicating the City in a Location-Based*

Performance [online]. Available from: http://www.amutualfriend.co.uk/papers/3.Uncle_Roy_at_ACE.pdf [Accessed 12 July 2007].

CHEOK, A.D., FONG, S.W., GOH, K.W., YANG, X., LIU, W. AND FARZBIZ, F., 2003. Human Pacman: a sensing-based mobile entertainment system with ubiquitous computing and tangible interaction. *In: Proceedings of the 2nd workshop on Network and system support for games in 2003 Redwood City, California.* New York: ACM Press, 106-117.

GIRARDIN, F., 2005. *Pervasive Game Development Today* [online]. Available from: http://www.girardin.org/fabien/catchbob/pervasive/ [Accessed 10 July 2007].

KNIGHT, W., 2004. *Human PacMan hits real city streets* [online]. Available from: http://www.newscientist.com/article.ns?id=dn6689 [Accessed 10 July 2007].

KOTAKU. *WiFi Plundr Coming to DS* [online]. Available from: http://kotaku.com/gaming/gps/wifi-plundr-coming-to-ds-266488.php [Accessed 15 July 2007].

Plundr [online]. Available from: http://plundr.playareacode.com/ [Accessed 15 July 2007].

SOLON, R., HAHN, R.M., BARBOSA, J., 2005. *BlueGame: Um Jogo Móvel Multijogador baseado em Comunicação bluetooth* [online] Universidade do Vale do Rio dos Sinos. Available from: http://www.inf.unisinos.br/~barbosa/textos/WJOGOS_2005.pdf [Accessed 25 June 2007].

What Went Wrong? A Survey of Problems in Game Development

Fábio Petrillo Marcelo Pimenta Francisco Trindade Carlos Dietrich
Institute of Informatics - Federal University of Rio Grande do Sul, Brazil

Abstract

Despite its exuberance and profitability, many reports about games projects show that their production is not a simple task, surrounded by common problems and being still distant from having a healthy and synergetic work process.

The goal of this paper is to present a survey of problems found in the development process of electronic games, collected mainly from game postmortems, exploring similarities and differences of well-known problems in the traditional industry of information systems.

Keywords: Electronic games, game development, problems in game development, survey, postmortems

Author's Contact:
fabio@petrillo.com
{mpimenta,fmtrindade,cadietrich}@inf.ufrgs.br

1 Introduction

The game development industry is entering a new age, in which technology and creativity walk together, producing some of the most astonishing entertainment activities of the 21st century. This industry has an annual income of billions of dollars, and in 2003 it exceeded its revenues comparing to the movie industry, it emerges as one of the most powerful, exciting and influential industry in the world of the arts [Gershenfeld et al. 2003]. Great game projects count on highly specialized multidisciplinary teams, having simultaneously software developers, designers, musicians, scriptwriters and many other professionals. Thus, the game developer career is currently one of most dynamic, creative, challenging and potentially lucrative that someone can have [Gershenfeld et al. 2003].

However, game development is an extremely complex activity [Gershenfeld et al. 2003] and a much harder task than one can initially imagine. To create a game is a different experience from what it was in 1994 and, certainly, much more difficult, having boosted in complexity in recent years [Blow 2004].

The main proposal of this work is to collect and discuss the problems that afflict the electronic games industry, because surprisingly very little attention has been paid to collect information about them, to organize this information in a systematic form and to quantify the frequency in which they occur.

The steps we have taken are:

- analyze the problems of the traditional software industry, collecting information to compare with the game industry;
- analyze the game industry problems in specialized literature, making it possible to map of these problems in the analysis of postmortems.
- collect, through postmortems analysis, focusing aspects of software engineering, the real problems cited by professionals of the game industry, making a qualitative analysis of the outcome;
- compare the results obtained in the game industry with the data of the traditional industry, searching for similarities and idiosyncrasies;

This paper is structured as follows: after this introduction, section 2 brings a study of the traditional software industry problems, presenting statistics of the projects performance and discussing the traditional problems of software engineering. In section 3 we discuss the problems cited in the specialized literature about game development and in the postmortems, numbering them and quoting examples that demonstrate its relevance in the project context. After having presented this situation, we discuss the methodology used in the analysis of postmortems and describe the results obtained in the problems survey. After that, a comparison between these results is made and the problems discussed in section 2.

Finally, section 4 discusses the conclusions and future work proposals.

2 Software Industry Problems

The analysis of problems in the development of software systems goes back to the origin of Software Engineering. The famous NATO conference in 1968 [NATO 1968], in which the term Software Enginnering [Sommerville 2001] was created, had as its basic goal the analyze of the key problems of critical areas in the process of programs construction.

Although the academic community has being discussing for almost 40 years the problems that afflict the development of computational systems, we still have difficulties in elaborating elegant and correct software, which satisfies customers and meets estimates, as much in time as in budget [Bach 1995; Pressman 2006], and the current status of software development can be seen as far from an ideal situation. Jones [1995] suggests that software can be faced today as one of the most problematic aspects of the corporative world. Costs and deadlines are not met; imperfections are more common in large software projects then in almost any other business area. Failures in projects happen in all countries, in big or small companies, in commercial or governmental organizations, not taking in account status or reputation [Charette 2005].

The research made by the Standish Group [1995] shows that more than 30% of the projects are cancelled before being completed and that 53% of the projects have a cost exceeded in 189%. Only 16% of the projects are completed in the stated period and budget, with all the initially specified functionalities [Standish Group 1995]. In this scenario, Sommerville [2001] suggests that the software engineering is not really in a crisis, but in a state of chronic agony.

With all these difficulties, many software projects fail and these failures are not reflected in changes. As cited in The Chaos Report [Standish Group 1995], many errors in software projects are not deeply investigated and the same errors continue to be committed. The corporative insanity existent in software development is inserted into the fact that the same things are repeated in different opportunities, and a different result is expected every time. For these reasons, DeMarco and Lister defend that the main problems of software projects are not technological, but managerial.

The Standish Group, in its research of the problems of software projects, quantified the main factors that cause damages or cancellations of projects. In this research, made with IT executives, the opinion requested was which would be the main factors that cause problems in the projects. Thus, 13.1% of the executives have cited incomplete requirements and 12.4% the lack of user envolvement. In section 3.5, we will use some of these results as support for the comparison between the traditional and the electronic games industry.

From these surveys and from the analysis of the references made by Yourdon [2003] and Charette [2005], it is possible to group the problems in four main categories: 1) schedule problems; 2) budget problems; 3) quality problems; 4) management and business related problems. In fact, IT projects rarely fail for only one or two reasons. More frequently a combination of these problems is found, being associated with one another [Charette 2005]. In the sections to follow, we will analyze each one of these problems, characterizing them and discussing their causes.

2.1 Schedule Problems

All the project that do not fulfill the foreseen deadlines has schedule problems. As cited by Brooks [1995], exceeding all the other causes together, the biggest cause of project cancellations is problems with schedule.

Perhaps the main reason for this problem is the weak estimate techniques, confusing effort with progress [Brooks 1995]. With problems in the estimates, the work progress is poorly monitored and when delays are detected, the natural answer is to add more professional help to the project.

In an article about the limits of estimate definition in software projects, Lewis [2001] comments that many programmers discover by themselves that it is impossible to make accurate estimates in software projects.

Another factor that leads to schedule problems is the optimism. Brooks [1995] claims that all programmers are optimistic. This optimism, followed by naivety, that frequently is associated with inexperience, causes people not to have conscience of how much time and effort will be necessary in the construction of a system [Yourdon 2003]. This situation is so common that DeMarco and Lister [2003] call it "hysterical optimism": everbody in the organization desperately wants to deliver a complex system, that would have never be completed in a realistic estimate of at least three years of effort, in 9 months.

Besides optimism, cowardice is another sociological aspect found during the estimate of projects. Brooks [1995] observes that programmers, as well as their bosses, are influenced by the sponsor's urgency during the elaboration of the project schedule. This urgency can even rule the period of a task, but it cannot rule the time that is necessary for its accomplishment. False schedules are produced to answer the sponsor's desire, and this unrealist position is much more common in software projects than in any other engineering.

The schedule problems due to the optimistic estimates, cowardice and uncertainties have caused damages to the credibility of the software industry. Lewis [2001] claims that the credibility will not be achieved while absolute deadlines continue to be promised. On the other hand, the software industry must acquire knowledge about the existing uncertainties and the intrinsic risks of software development, being necessary to promote a public and honest discussion about these risks.

2.2 Budget Problems

Projects with budget problems are characterized by demanding investments above the estimated. Models to estimate the cost of a project are based on function of the necessary development time or measures of size, as lines of code or points of function [Lewis 2001]. Consequently, the cost of a product is also a function of the number of professionals needed for its development [Brooks 1995], as men/month or men/hour.

On average, the budget of software projects exceed in almost 200% the original estimated cost, phenomenon that happens in companies of all sizes [Standish Group 1995]. The study made by the Standish Group [1995] shows that some projects (4.4%) exceed their budget in more than 400%.

2.3 Quality Problems

A project has quality problems, in the customer's point of view, when there are differences between the final product and the expectation of it. The quality is associated to the customer's satisfaction, aesthetic aspects and the correction of a system in terms of its requirements [Bach 1995].

In the technical point of view, the most common reason for disasters in software projects is the poor quality control. According to Jones, to find and to correct bugs are the most time consuming and expensive aspects of software development, especially for large systems.

Moreover, according to the Standish Group [1995], of all the projects with problems, one quarter was delivered with 25% to 49% of the originally specified functionalities. On average, 61% of the originally specified functionalities are present in the end product.

2.4 Management Problems

Management problems happen in several ways, including bad communication; lack of investment in the team's training; and the lack of project inspection in regular intervals [Charette 2005]. According to Jones [1995], bad management of projects is the origin of many problems in the process of software development.

The project decisions are frequently difficult to be taken because they involve choices based on incomplete knowledge, turning the estimate of how much a project will cost or how much time it will need into much more of an art than a science [Charette 2005].

3 The Electronic Games Industry Problems

In this section we will discuss the problems of the electronic games industry. We will initially analyze what postmortems are and will show the problems which are cited by the game development specialized literature, demonstrating its characteristics and pointing out the criteria that were used to define this situations as problems. Finally, we will compare the collected results with the data obtained in the traditional software industry.

3.1 The Game *Postmortems*

The term *postmortem* designates a document which summarizes the project development experiences, with a strong emphasis on positive and negative aspects of the development cycle [Hamann 2003]. It is commonly done right after the project finishes, by managers or senior project participants [Callele et al. 2005]. In a software engineering viewpoint, postmortems are important tools for knowledge management [Birk et al. 2002], from which the group can learn from its own experiences and plan future projects. In fact, the postmortem analysis can be so revealing that some authors [Birk et al. 2002] argue that any project should be finished without it's own postmortem.

Postmortems are much used in the game industry. Many game websites devote entire sections to present these documents, such as *Gamasutra* (`http://www.gamasutra.com`) and *Gamedev* (`http://www.gamedev.net`). It is also very interesting to note the variety of development teams profiles and projects behind these documents, varying from few developers in small projects to dozens of developers in five-year-long projects.

The postmortems published by *Gamasutra* mainly follow the structure proposed by the *Open Letter Template* [Myllyaho et al. 2004], which is composed by three sections. The first section summarizes the project and presents some important aspects of the development. The next two sections, however, discuss the most interesting aspects to the game developers:

- **What went right:** if discusses the *best practices* adopted by developers, solutions, improvements, and project management decisions that have improved the efficiency of the team. All these aspects are critical elements to be used in planning future projects.
- **What went wrong:** it discusses difficulties, pitfalls, and mistakes experienced by the development team in the project, in both technical and management aspects.

The postmortem is closed with a final message from the author, commonly followed by a project technical brief, which includes the number of full- and partial-time developers, length of development, release date, target platforms, and the hardware and software used in the development.

The information contained in the postmortems constitute knowledge base that can be reused by any development team, which in-

cludes examples and real life development experiences. They can help in knowledge sharing, and can be very useful for planning future projects.

3.2 Problems found in specialized literature

According to Flood [2003], all game development postmortems say the same things: the project was delivered behind schedule; it contained many defects; the functionalities were not the ones that had originally been projected; a lot of pressure and an immense amount of development hours in order to complete the project.

The specialized literature in electronic games development is also effective in describing the problems of this industry. For Bethke [2003], the electronic games industry adopts, in general, a poor methodology, if any, causing projects to have longer duration in comparison to what they were suposed to, excedding the budget and tending to be irrationally full of defects.

This scenario is responsible for declarations as this, of Ian Lane, of Mad Doc Software [Gershenfeld et al. 2003]:

"It's rare a project where there is not some form of agony. I am trying to think of one that wasn't miserable in some way. I can think of one that was very smooth, but in general, I think you'll find that there is always some degree og angst associated with making games."

and of Scott Campbell, president and founder of Incognito Studio[Gershenfeld et al. 2003]:

"Every projects has a moment where you are totally in the hell, especially when you are making new intellectual property or trying to create the "fun" and core game identity."

In order to elucidate these problems in a better way, we will discuss some of them at great length, based on the causes of imperfection referred by Flynt and Salem [2004] and through the study of the electronic games development specialized literature.

3.2.1 Scope problems

According to Flynt and Salem [2004], the biggest reason of games project imperfection is the failure in clearly establishing the project scope. If a project does not have a well established target, emergent requirements can cause significant structural changes in the system's architecture, causing serious problems [Callele et al. 2005].

The development teams lose themselves in the scope when some difficulties arise: problems with the exaggerated size and complexity of the project, in addition to facing highly specific requirements of the games domain [Gershenfeld et al. 2003; Blow 2004]. However, the main cause of scope problems is the common situation where new functionalities are added during the development phase, increasing the project's size. This practice is known in the industry of games as feature creep.

The late addition of features occurs when developer, managers or customers add requirements to the project after its scope has already been defined. With the intention to create the best possible game, there is the habit of adding more elements everytime, creating perhaps a more interesting game (even without any guarantee) and certainly bigger [Gershenfeld et al. 2003].

This good intention results in features addition due to many reasons. According to Flynt and Salem [2004], the most common form in the addition of new elements occurs when programmers, by chance, discover interesting features during the development process or when they randomly decide to add functionalities that are considered attractive to the game. Flynt and Salem [2004] defend that the reason for this situation is in the development phase initiation without a strong effort of analysis, in addition to the lack of techniques to improve the investigation and verification of the project's requirements.

Another common form of feature creep happens when an external code (a new component, for example) happens to be incorporated without planning, for the reason or purpose of gaining time. However, what happens many times is that an extremely hard work is necessary for the integration of this new component. One third form takes place when developers decide, despite counting on a set of solid libraries, to implement their own algorithms.

However, the games industry has a lot of examples of situations in which features discovered during the development phase have transformed the game into a success. Game development is not a linear process [Flynt and Salem 2004]. Thus, if an interesting function is discovered, it must be analyzed in terms of its risks and, if viable, be added to the project schedule.

3.2.2 Schedule problems

The electronic games specialized literature exhaustingly describes schedule problems in game projects. Although they usually initiate with reasonably structured schedules, bright ideas and enthusiasm, something always goes wrong. And the number of things that can go wrong is virtually endless [Bethke 2003]. These problems are many times associated with the multidisciplinarity needed in the elaboration of games, generating delays caused by waiting for the work of other team members, in addition to the lack of a realistic estimate on the initial plan of development, making the team not capable of finding a deadline for the projects [Flynt and Salem 2004].

The production of estimates of time needed to complete a task has a series of risks that imply in underestimates, causing cumulative schedule delays. According to Flynt and Salem [2004], developers recurrently fail in their estimates due to lack of historical data that should assist the perception of time needed to carry through a task. Another common problem in deadline estimate is not taking into account the time consumed in complementary meetings and other activities, in addition to not planning the time needed to correct defects in the game.

However, for Flynt and Salem [2004], the key to the problems in the project's schedule is the unbalance among quality, cost and opportunities. If a game delivered two months after the release of a similar game from another company, the possibilities for success are diminished. On the other hand, a game that is launched before another one with an important number of defects, can have more problems than one that is delivered later with less bugs.

3.2.3 *Crunch Time*

In basketball, crunch time is the term that defines the last minutes of the game, when both teams fight to win the victory. It was invented in New Zealand in 2004, from the combination of the phrases *"show time"* and *"when it comes to the crunch"* [Wikipedia 2007].

In the games industry, the term is used for the periods of extreme work overload, typically occurring in the last weeks before validation phase and mainly in the weeks that precede the final deadline for the project delivery. In these periods, a work load of more than 12 hours a day is common, from 6 to 7 days per week, without rest intervals.

Although this cyclical problem is also present in traditional software companies, Gershenfeld et al. [2003] claims that the electronic games industry goes more frequency through crunch time periods. This assertion can be confirmed by Hamann [2003], when describing his experience in a project:

"On one game I worked on, one of the team members complained to the producer that months of working 12-hours-per-day, seven-days-a-week was getting to him. He said he'd like to have at least one weekend off. The producer replied, "This is the game industry - crunch time is a fact of life." I have a question for all of you game company owners, producers and leads out there: Ever wonder why you have such a high turnover rate at your company? Studies have shown that most game developers will put up with a severe crunch mode for one, or at most two, projects and then they start looking for another company. Believing that crunch time is a fact of life is a result of following the "blind religion" that's so pervasive in our industry".

Gershenfeld et al. [2003] comment that periods of crunch time can be good for single or ambitious people, who make of their work the main aspect of their lives. However, they claim that this situation is not sustainable for people who have a family or want to keep a serious relationship. Therefore, they suggest the construction of schedules that contemplate a healthy pace of daily work, defending that people can be much more efficient if they work in a typical day of 8 hours.

3.2.4 Technological problems

As a software device, all games are technology dependents. Moreover, the technologies used in games are so advanced that the leaders in the area of graphical computation are games companies. However, the technological component brings risks to the projects of games. Flynt and Salem [2004] describe the risk of using new technologies, that many times cause a great effort and high investment of time in order to use them.

According to Gershenfeld et al.[2003], the technological risks are generally higher when the team is working in a new platform, that has not been completely delivered neither consolidated. The first one of these risks is that no developer has ever worked in it. The second risk is that in many times the platform hardware still contains problems, that are only found by development team of the *launch title* [1].

3.3 Problems found in the postmortems

Although specialized literature in development of games contain valuable descriptions concerning the problems of this industry, as we saw in section 3.2, another way to raise the problems is through *postmortems* reading. Its reading is deeper (forceful and specific, discussing the problems more deeply) and more accomplishing about the problems in specialized literature.

With the objective of raising the problems of electronic game industry, 20 postmortems published in the site Gamasutra were analyzed. The analysis process occurred through the detailed reading of stories, extracting the ideas that demonstrate the problems faced by the teams during the game development process.

Therefore, we will describe the main problems found in postmortems, exemplifying it through quotes, aiming at defining and exemplifying the adopted criteria of analysis of the described survey in section 3.4. In this work, 15 problems were analyzed, these are the most relevant and recurrent found in postmortems.

3.3.1 Unreal or ambitious scope

It is not surprising that many postmortems report explicitly as having an unreal or extremely ambitious scope. The *Gabriel Knight 3* project [Bilas 2000], for example, brings the following sentence: *"... was an extremely ambitious project combined with an extremely inexperienced team"*.

Some projects as *Command and Conquer: Tiberian Sun* [Stojsavljevic 2000], *Vampire: The Masquerade* [Huebner 2000] and *The X-Files* [VandenBerghe 1999] dedicate a complete section to report this problem. Huebner [2000] comments that *"... Many of the team members had wanted for some time to do a really huge, ambitious role-playing game"*.

Stojsavljevic [2000] describes in his section *"Unrealistic expectations"* that *"The degree of hype and expectations that Tiberian Sun had to fulfill was staggering"*. Moreover, in the project *Black & White*, Molyneux [2001] uses the expression "insanely ambitious" in his briefing: *"... looking back, I don't know whether we were insanely ambitious, because at the time we started, you couldn't have done what we did"*.

[1] *Launch title* is a game that is delivered at the same time that the new platform.

3.3.2 Feature creep

An interesting pattern found in many game projects is the feature creep, that is, as new modules are added without planning, during software construction. This problem is literally described in the project *Vampire* [Huebner 2000]:

"New engine features get added too late in the schedule to be utilized fully by the designers and artists. This happened several times during Vampire. Some of the more interesting special effects, for example, were added only a few weeks before the data was to be locked down for final testing".

This could also be see clearly in citations made by Reinhart [2000] for *Unreal Tournament* project:

"... The amount of content grew and we soon realized we had a much larger project on our hands than we had originally thought. ... Unreal Tournament is a very fun game with a lot of features packed into a short amount of development time. Those features were largely added through spur-of-the-moment decisions".

In many postmortems phrases are found as: *"As we realized this late in the project..."* [Smith 2001]; *Humans procrastinate Humans. ...Audio was plugged in at the last moment..."* [Scheib 2001] or *"we were capable to incorporate several features that originally had not been planned to be developed"*.

In the *Draconus* project, Fristrom [2000] report his experience with this problem:

"Another example of poor planning was that magic was an afterthought. We always knew we would have spells in the game, special abilities with nice special effects that your character would acquire as the game progressed, but they were left until late in development because there were always more pressing issues. Of course, once we did implement spells, they unbalanced the play of the game horrendously, and much more unplanned work was needed to make magic function reasonably".

3.3.3 Cutting features during development process

As well as the feature creep, another problem frequently quoted is the cut features during development process. As for the fact that projects initiate in an ambitiously, many functionalities are imagined and even implemented, but for diverse factors they end up being further on in the project.

The typical case is found in the *Rangers Lead the Way* [Ridgway 2000], in which the initial features of the game were modified to hold the schedule and the budget:

"We dropped the Macintosh version right away, and we terminated the PlayStation version at alpha (when it was still running at five to eight FPS). Networking support was postponed for an expansion pack a year into development"

Or still reported by Fristrom [2000]:

"Our answer to being under funded and under scheduled was to cut network play. We needed to make much bigger cuts".

Another example of this situation is associated with the unreal scope. The report made by Malenfant [2000] for the game *Wild 9* represents this situation well:

"... The material they came up with is enough to fill up three or four games like this one. In addition, the whole team took this game to heart and ended up coming up with suggestions of their own. We were thus faced with a situation toward the end of the project: we could not fit everything into the game".

In the *Tropico* [Smith 2001], this problem was caused by a different reason: lack of physical space demanded to cut features. Smith [2001] comments this situation: *"... we had to cut other features to create space, features which would have improved the game"*.

3.3.4 Problems in the design phase

Although designing is a common practice in the game projects, mainly by design document or game bible, many reports claim that there are problems in this phase. The postmortem of *Wild 9* brings an emblematic testimony:

"Many times, titles are late because certain gaps in the original design were overlooked and the full design was never really laid down on paper before development started".

Barrett et al. [2002] dedicate an entire section for this problem, titled *"Design on the fly"*:

"... Because we were designing a game to the technology (rather than the other way around), we were throwing out design documents as quickly as they could be written. Art assets had to be revised, retextured, discarded, and rebuilt from scratch several times. As most readers will know from experience, this is a scenario for feature creep, obsolete tool sets, and blown deadlines".

On the other hand, Saladino [1999] says that the opposite situation may also be problem:

"One interesting example of the opposite problem arose in our custom interface system: we had too much design. Our interface system is the library of code that handles our front end interface and our in-game heads up display. The code was written by a programmer with extensive experience in C++ object oriented design and loved to use it. He spent many weeks designing and crafting an interface system which was extremely... well... designed. Private data fields were placed into private headers. The "const" operator was used everywhere. Void pointers hiding internal data structures made debugging a nightmare".

Moreover, Saladino [1999] makes the following recommendation:

"... There are extremes at both ends of the design spectrum and either one can seriously damage overall productivity. Make sure that you spend time designing what is important".

3.3.5 Delay or optimistic schedule

Many game projects quote delay schedule. For example, Ridgway [2000] describes this problem clearly in the *Rangers Lead the Way* projetc: *"We learned the hard way what's possible to accomplish in 15 months; as a result, we completed the game in 20"*.

Ragaini [2002] analyzes deeply this problem, describing its main cause:

"Depending on how far back you look at the schedules, Asheron's Call was either one to two years late. ... Deadlines were consistently missed. A lot of this was due simply to underestimating the time required for development tasks. This created a domino effect as we continually played catch-up, trying desperately to make up for lost time".

In the same way, VandenBerghe [1999] describes this problem literally: *"The most severe blow suffered by all teams was from accepting an unrealistic schedule"*. Again, optimism is the cause of problem, that is shows in Fristrom [2000] testimony for the *Draconus* project: *"In my experience, publishers love overoptimistic schedules"*.

3.3.6 Technological problems

The problems of failure in third party Application Programming Interface (API), on platform or hardware were classified as technological problems. A typical example was faced in the *Age of Empires II: The Age of Kings* [Pritchard 2000] project:

"Microsoft's DirectPlay API still has a number of issues that make it less than perfect. One of its biggest problems is documentation and testing of the lesser-used portions of the API, or rather the lack thereof. Late in the development of AoK, our communications programmer, Paul Bettner, was able to communicate with the DirectPlay developers and an interesting scenario played out several times: Paul would attempt to solve some problem and the developers would indicate that it wouldn't work because of bugs in DirectPlay that they knew about but that were not documented".

However, prosaic problems also are found, as occurred in the *Draconus* project [Fristrom 2000], with the C compiler: *"We lost many weeks in our battle to find a compiler that could actually compile our code"*.

The use of external components from third parties many times, in the end, produce more difficulties in development process.

"Our external solutions for rendering and networking both fell through and had to be replaced with internally developed code late in the development cycle. ... In retrospect, we would have saved money and had a much smoother development process if we'd bitten the bullet early on and committed ourselves to building our own technology base".

3.3.7 Crunch Time

As discussed in section 3.2.3, crunch time was found in many postmortems. Perhaps the most eloquent testimony about this problem has been made by VandenBerghe for the *The X-Files* project:

"The most severe blow suffered by all teams was from accepting an unrealistic schedule. Despite endemic problems, ... the concept that was floated at the time was that it would be possible to adhere to the original schedule if everyone simply worked around the clock. Foolish and naïve, we bought it, and started pushing. While a final push of a few weeks or even a few months is common toward the end of a technical project, the schedule of six to seven days a week of twelve- to twenty-hour days adopted by The X-Files team stretched on for eight months. Predictably, exhaustion began degrading the ability of people to produce quality work, not to mention the deeply straining effects it had personally on team members and their families. In retrospect, the death march did little to speed the project's finish. There were several systems that were developed during this time that caused us no end of grief in development, largely because of the inability of the developers to make clear and rational design decisions from lack of sleep. ... For me, the lesson here is that if the schedule begins to slip, the solution is not as simple as just applying more work to the problem. Pushing harder won't necessarily finish the project any sooner, and some of the damage done to the team under that much stress may be irreparable".

The sleep deprivation was also cited by Spanel e Spanel [2001] for the *Operation Flashpoint* [Spanel and Spanel 2001] project:

"After a long, sleepless night of playing through the game and fixing any problems that appeared, everything looked fine, and most of the team could finally go to sleep again".

Moreover, the *Diablo* [Schaefer 2000] project showed that passed experiences not necessarily produce best practices for new projects:

"The original Diablo went gold on the day after Christmas in 1996, after a grueling four-month crunch period. We hadn't put any thought into what game to do next, but as most developers can probably relate to, we were pretty certain we weren't ready to return to the Diablo world after such a long development cycle. The only thing we were certain of was that we wanted to avoid another crunch like we had just experienced. Diablo II went gold on June 15, 2000, after a grueling 12-month crunch period".

3.3.8 Lack of Documentation

Although many projects report sucess on documentation processes, mainly with the project document, many postmortems describe congenital problems by lack of documentation, as was describes by Spanel e Spanel [2001] in *Operation Flashpoint*:

"Lack of documentation is a common affliction among game developers, but some aspects of this problem were so severe in our case that they are worth mentioning. While we'd never believed too much in designing the game on paper, the real problem was that we never even had documentation of the things that we'd finished. This

situation led to incredible problems in the final stages of development. Many tasks could only be done by one person on the whole team. In other cases, hours were spent trying to investigate how something had originally been meant to work. We recognized these problems and tried to improve them, but apart from a few instances, our effort wasn't really successful. As the development team grew, the missing documentation was becoming a more serious problem. But the final project deadlines were getting closer as well, so it was nearly impossible to find the time to address the problem".

Even not having made a project document, Reinhart [2000] defends its utility:

"If we develop a design document, we'll use it with the understanding that it can be modified at any time. That having been said, I think there is a definite positive argument for having some sort of central guide to everyone's ideas. Having the ability to sit down and look over the big picture is very valuable".

3.3.9 Communication problems

Communication problems are also described in postmortems, manly between tecnical and art team, as report Fristrom [2000] for *Draconus* project: *"The game designers and artists didn't really communicate as well as they should have on the levels they made"*.

Another interesting situation was the report about communication problems between the tecnical team and the publisher during quality control [Ragaini 2002]:

"Communication between Microsoft and Turbine was also a major factor. The teams were separated by about 3,000 miles and three time zones. Although weekly conference calls were scheduled, they lacked the collaborative mentality necessary for maintaining a successful relationship. E-mail threads were either ignored or else escalated into tense phone calls, and in some cases the bug-tracking database (RAID) was not used effectively. Clearly, everyone would have benefited from more face-to-face time. E-mail - and even conference calls - are poor media for managing new and sensitive corporate relationships, especially ones between companies with such different corporate cultures. From a developer's perspective, it's always easy to blame the publisher for unrealistic expectations and bureaucracy. What's important to realize is that it is everyone's obligation to communicate expectations and problems before they escalate to the point of being a crisis".

3.3.10 Tools problems

In the game elaboration, tools are essential elements in the construction process, allowing to elaborate parts of the game, project control or generate a version to deliver.

The *Diablo II* project had problemas with its tools [Schaefer 2000]:

"The greatest deficiency of our tools was that they did not operate within our game engine. We should have made tools that let us create content within the game engine".

Fristrom [2000] describes dramatically how the lack of effective tools can be problematic: *"The process for making a release build was insane"*.

The tools problems in some projects were related to version control process, as was described in *The X-Files* [VandenBerghe 1999]:

"We used SourceSafe 4.0 for our source control. It worked effectively as a code database, but was a less effective choice for a revision control system. It gave us one nasty shock during the PSX port: we lost all of our revision history due to being unable to move the revision database from one server volume to another (larger) one, but fortunately this didn't end up hurting the project much".

3.3.11 Test Problems

The test problems were identified in some projects, being that the postmortem most explicit was made by Bernstein [2002]: *"If we had done more beta testing, with a larger group and earlier on, we would have gotten the kind of outside feedback that would have helped us realize that some of the tradeoffs we were making were going the wrong way"*; and by Upton [2000] for the game *game Rainbow Six*:

"We got lucky. As a result of our early missteps, the only way we could get the game done on time was to cut deeply into our testing schedule. We were still finding new crash bugs a week before gold master; if any of these had required major reengineering to fix, we would have been in deep trouble."

The insufficience in the test phase was identified by Meynink [2000] also as a problem associated with the publishers:

"Never underestimate your testing requirements and, more importantly, never let your publisher underestimate your testing requirements. Almost obvious to anyone who has developed a game should be the testing phase, especially for projects with shorter deadlines. Make sure that the level of testing support you expect is written into the contract with your publisher. ... In the future, we will be sure to have a testing plan explicitly stated in both the contract and the development schedule".

3.3.12 Team building

Some postmortems describe problems found in the team building, causing communication and relationship problems. An example, was happening in Wild 9 [Malenfant 2000], in which the project has 2 different teams:

"The first Wild 9 team was a great collection of talented individuals, while the second was a talented team. This statement alone sums up the main problem encountered by the first team. As we found out later in the project, communication and good atmosphere were two factors that made this project move forward; it's now obvious that the best thing to do was to start over rather than try to salvage a situation that was going nowhere".

Another unusual situation was described by Ridgway [2000] for the *Rangers Lead the Way*:

"The last thing that I expected was that it would be hard to find good programmers in Seattle. I was hired at the start of SpecOps' development and didn't manage to bring the entire programming staff on board for nearly seven months. Needless to say, this caused substantial delays. We had similar problems hiring the art team".

As an essential characteristic in the game development is the experience of the team. Ragaini [2002] describes the making up of his team and the effect of his choices:

"None of the senior developers at Turbine (including me) had ever shipped a retail PC game. None. Many of the employees were students immediately out of college, or even college students completing a work-study program. This obviously was the source of several severe problems in the development of Asheron's Call".

3.3.13 Great number of defects

A characteristic of some projects was the great number of defects found in some development phase, being that the *Draconus* [Fristrom 2000] project tells this problem explicitly: *"The bug list went over three thousand"*.

However, the most maked testimony about this problem was made by Molyneux for the *Black & White* game:

"After canceling our Christmas party on December 26, 2000, we managed to hit Alpha, which as any developer knows is a very loose definition, but at least we could say that all the game features were now locked. After a well-deserved Christmas break, we came back to find that we had more than 3,000 bugs. We had six weeks to reduce this to zero, but the thing about bug-fixing is that you can solve one problem but in doing so create three more. So although we worked as hard as we could, the overall figure crept down slowly rather than dropped at the rate at which we were actually sorting out the bugs. ... The irony was that the last 10 bugs were the hardest to fix, and with every one there were four more created. It was as if the game just didn't want to be finished and perfected".

3.3.14 Loss of Professionals

Some comment that problems that affected seriously their projects was the loss of professionals. It is that what Upton [2000] tells in his *Rainbow Six* project:

"Losing even a junior member of a development team close to gold master can be devastating. When our lead engineer took ill in February 1998, we were faced with a serious crisis".

This problem also afflicted the *Rangers Lead the Way* [Ridgway 2000] project, but this case the problem was in the art team:

"About a month before E3 '97 rolled around, our art lead and a senior artist decided to leave the company. I had designed most of Viper's capabilities with them, and losing them at that critical time really devastated the project".

3.3.15 Over Budget

The over budget was the less cited problem, being that only 2 projects explicitly referred to it: *"... about $400,000 more than we has budgeted to tackle it"* [Fristrom 2000].

However, it was the *Gabriel Knight 3* project that had this problem more highlighted, since original estimate was well less than US$ 2 million, but the final budget finished in US$ 4.2 million [Bilas 2000].

3.4 Analysis of postmortems and research results

The electronic games industry, for its competitiveness and corporative way, generally turns inaccessible its internal data of projects for researchers [Callele et al. 2005]. For skirting their difficulty, the postmortem analysis described in the section 3.1 was used, as observation source.

The 20 postmortems analyzed in these survey are listed in the table 1. Based on it, it is possible to perceive that projects of several team size, budget and development time were analyzed. It is also important to point out that all postmortems analyzed ware finished projects, that is, projects that had resulted in complete games and delivered to the market.

Project	Team	Development Time (Months)
Beam Runner Hyper Cross	5	4
Dark Age of Camelot	25	18
Black & Whire	25	37
Rangers Lead the Way	13	20
Wild 9	18	NI
Trade Empires	9	15
Rainbow Six	22	24
The X-Files	36	48
Draconus	19	10
Cel Damage	16	24
Comand and Conquer: Tiberian Sun	35	36
Asheron's Call Massive Multiplayer	60	48
Age of Empires II: The Age of Kings	40	24
Diablo II	40	36
Operation Flashpoint	10	48
Hidden Evil	22	12
Resident Evil 2	9	12
Vampire: The Masquerade	12	24
Unreal Tournament	16	18
Tropico	10	12

Table 1: *Analysed postmortems*

The survey process was took place in 3 stages. In the first stage, each postmortem was read and citations that were found interesting ware highlighted. In the second, using the survey made in section 3.2, the problems were classified to be tabulated. In the third stage, each postmortem again was read, with special attention to the highlighted sentences, tabulating each sentence according to classification made previously. The final result of this tabulation produced table 2 (see appendix).

In this table, the columns were reserved for the problems and the games were arranged in lines. For each postmortem was executed the study in each one of the 15 problems discussed in section 3.3. Each problem found was marked with a "Yes". This value was only attributed to the **explicitly** mentioned problems by the author, as it is possible to identify in the described examples in section 3.3. For the problems that were not mentioned by the author or explicitly cited that had not occurred, a "No" was attributed. It is important to mention that if this valua as attributed to a certain problem it does not necessarily mean, that a problem did not happen, but that it may have been considered irrelevant for the postmortem author during the elaboration of his testimony.

So that we can in some way quantify the problems found in postmortems, the number of occurrences of "Yes" was counted in lines and columns. The amount of "Yes" found in the lines represents how many different types of problems a certain game presented. On the other hand, the amount counted in the columns represents the number of occurrences of this problem in the set of analyzed games. The accounting of the columns, that can be seen in the penultimate line, propitiated that the table was organized in sequence decreasing of occurrences. The last line contains the percentage of occurrences in relation to the number of studied projects.

When we analyze this results more closely, we see that the most cited problems were the **unreal or ambitious scope** and **features creep** with 75% (15 of 20) of projects reporting this problems. After that, demonstrating the coherence of the postmortems, 70% of projects citing the **cutting features during development process**. The other most found were **problems in the design phase** and **delay or optimistic schedule**, with 65%. Also the **technological problems**, with 60% can be highlighted. Figure 1 presents the histogram of occurrence of problems in sequence decreasing, in which we can make a graphical comparison of these results.

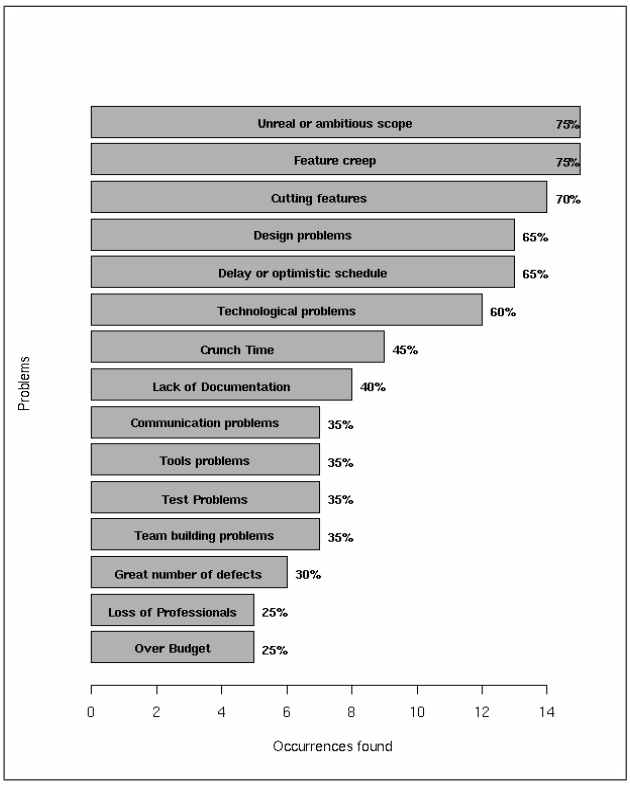


Figure 1: *Occurrence of problems*

Perhaps the most surprising result was that problems said to be "universal" in the game, as **crunch time** and **over budget**, had an occurrence rate relatively low, being that for the **crunch time**, only 45% of projects showed this problem. The **over budget** and **loss of professionals** problems had the lower incidence among all the analyzed problems, only 25%.

3.5 Comparative between the traditional and electronic games industry

In last instance, to develop games is to develop software [Bethke 2003; Gershenfeld et al. 2003]. The statement also insinuates that it is possible to compare, into terms of its problems, the game development with the traditional software industry. In order to elaborate this comparison, we can use the problems survey made in the section 2 and in the results of the research done in the sections 3.2 and 3.4.

If we compare initially these two studies, we will see that in fact **all the main problems of the traditional software industry is also found in the eletronic games industry**. In both studies, the unreal scope is highlighted, as well as the problems with the requirements analysis. Also Charette [2005] points out that the unreal objectives and undefined requirements are among the main problems, strengthening the similarities.

But of all the problems, what really is almost universal, as much as in information systems rather than in projects of games, is the unreal scope or the exaggerated optimism. With a percentage of 75% of the projects of analyzed games and clearly highlighted by Yourdon [2003], Brooks [1995] and DeMarco and Lister [2003], although it is not surprising, show clearly that optimism and naivety in the scope definition, as well as in the estimate of effort needed to do a tasks, are determinant factors in the occurrence of most problems in the two industries. Moreover, both industries suffer from the Machiavellian attitude of controlling seniors and game publishers. For these reasons, as much as the traditional industry rather than games do not suffer essentially from technological problems, but management problems.

In terms of differences between the two industries, perhaps an important problem, specific to the game industry, is the communication among the teams [Callele et al. 2005]. In the traditional software engineering, the team is usualy relatively homogeneous, basically formed by technician with skills on information technology. However, in the electronic games industry, for been an activity eminently multidisciplined, it adds people with distinct profiles, as plastic artists, musicians, scriptwriters and software engineers. This mixture, in spite of been positive in the sense of having the more creative work environment, seems to produce a true split on the team, dividing it in to "the artists" and "the programmers" [Callele et al. 2005]. This division, that basically does not exist in the traditional software industry, causes important communication problems [Flynt and Salem 2004], since both teams believe to communicate clearly when using their specific terms to express their ideas. This is an important source of misunderstanding [Callele et al. 2005].

Another important diference is that the game requirements elaboration is much more complex, therefore subjective elements as the "diversion" does not have on efficient methods for its determination, the point of Callele et al. [2005] to identify clearly that it is necessary to extend the traditional techniques of requirement engineering to support the creative process of the electronic game development.

4 Conclusions and Future Work

From the research done with the postmortems analysis focusing aspects of software engineering, we see that the most cited problems have been the **unreal or ambitious scope** and **features creep** with **75% (15 in 20)** of the projects complaining about these two problems, followed by the **cutting of features during the development phase**, with 70% of the projects citing it and in third place, the **problems in the design phase** and **delay or optimism in the project schedule**, with 65%. The technological problems, with 60% can also be cited.

The study of postmortems shows that, the **unreal scope or the exagerated optimism**, mainly in the projects effort estimate, are probably important factors in the occurrence of the majority of problems. This situation can be seen as almost universal, being found as much in information systems as in game projects, with a 75% percentage of ocurrence in the analyzed game projects and clearly pointed by Yourdon [2003], Brooks [1995] and DeMarco and Lister [2003]. From these results, it is possible to conclud that the traditional and game software industry do not suffer essentially from technological problems, but from management problems.

In fact, **all the main problems of traditional software industry are also found in the games industry**, it is possible to conclude that they are related. In both studies, the unreal scope was pointed out, as the problems with requirements definition.

The so called universal problems of the games industry, like **crunch time** and **over budget**, have had a relatively **low** occurrence rate, with 45% for crunch time. The problems with **over budget** and the **loss of professionals** have had the lowest incidence among all the problems analyzed, with a ocurrence rate of only **25%**.

Comparing the two industries, we can perceive similarities in the frequency of some problems, as schedule delays and requirement problems, having almost equal rates. Budget problems have a considerable discrepancy, probably do to the greater emphasis in the technological and management aspects given by the stories authors, minimizing the financial aspects of the project.

The main limitations that were found are:

- Although relevant, all analyzed postmortems have been created after the conclusion of projects, with at least a game delivered. Cancelled projects have not been analyzed, neither projects that, despite the efforts, have not produced a complete game for the market. This can have led to optimistical results in comparison with what reality shows.
- The fact that postmortems do not count on a formal structure in its elaboration, being formed basically of stories in dayly language: the analysis of problems depended significantly on the text interpretation made by the researchers. Thus, different criteria of judgment could be adopted by different analyzers. Trying to minimize this problem, a set of searched criteria was used as a base for the analysis.

The results obtained in this work, as well as its limitations, excite some possibilities for future works:

- Since it focused on problems, it is possible to perform the same study aiming at the best sofware development practices which are applied in the games industry. Maybe we can surprise ourselves again, but now with the number of good practices adopted.
- An interesting work could be to attend the demand made by Flynt and Salem [2004], developing specific techniques to improve the inquiry and verification of requirements in electronic games projects.
- Listening to the recommendations of Callele et al. [2005], to consider a software enginnering methodology specialized in the domain of the games industry, contemplating aspects as complexity, multidisciplinarity and creativity.

References

BACH, J. 1995. The challenge of "good enough" software. *American Programmer Magazine* (October).

BARRETT, K., HARLEY, J., HILMER, R., POSNER, D., SNYDER, G., AND WU, D., 2002. Postmortem: Cel damage. Gamasutra - The Art & Business of Making Games, February. Available online at `http://www.gamasutra.com/features/20020227/wu_01.htm`.

BERNSTEIN, R., 2002. Postmortem: Trade empires. Gamasutra - The Art & Business of Making Games, January. Available online at `http://www.gamasutra.com/features/20020125/bernstein_01.htm`.

BETHKE, E. 2003. *Game development and production*. Wordware Publishing, Plano.

BILAS, S., 2000. Postmortem: Gabriel knight 3. Gamasutra - The Art & Business of Making Games, October. Available online at http://www.gamasutra.com/features/20001011/bilas_01.htm.

BIRK, A., DINGSOYR, T., AND STALHANE, T. 2002. Postmortem: never leave a project without it. *IEEE Software 19*, 3 (May), 43–45.

BLOW, J. 2004. Game development: Harder than you think. *ACM Press Queue 1*, 10 (February), 28–37.

BROOKS, F. P. 1995. *The Mythical Man-Month - Essays on Software Engineering*, anniversary edition ed. Addison-Wesley Professional, United States of America.

CALLELE, D., NEUFELD, E., AND SCHNEIDER, K. 2005. Requirements engineering and the creative process in the video game industry. In *13th IEEE International Conference on Requirements Engineering*.

CHARETTE, R. N. 2005. Why software fails. *IEEE Spectrum 42*, 9 (September), 42–49.

DEMARCO, T., AND LISTER, T. 2003. *Waltzing with bears - managing risk on software projetcs*. Dorset House Publishing.

FLOOD, K., 2003. Game unified process. GameDev.Net, May. Available online at http://www.gamedev.net/reference/articles/article1940.asp.

FLYNT, J. P., AND SALEM, O. 2004. *Software Engineering for Game Developers*, 1ª ed. ed. Software Engineering Series. Course Technology PTR, November.

FRISTROM, J., 2000. Postmortem: Draconus. Gamasutra - The Art & Business of Making Games, August. Available online at http://www.gamasutra.com/20000814/fristrom_01.htm.

GERSHENFELD, A., LOPARCO, M., AND BARAJAS, C. 2003. *Game plan: the insider's guide to breaking in and succeeding in the computer and vieo game business*. St. Martin' s Griffin Press, New York.

HAMANN, W., 2003. Goodbye postmortems, hello critical stage analysis. Gamasutra - The Art & Business of Making Games, July. Available online at http://www.gamasutra.com/resource_guide/20030714/hamann_pfv.htm.

HUEBNER, R., 2000. Postmortem of nihilistic software's vampire: The masquerade – redemption. Gamasutra - The Art & Business of Making Games, August. Available online at http://www.gamasutra.com/features/20000802/huebner_01.htm.

JONES, C. 1995. Patterns of large software systems: failure and success. *IEEE Computer 28*, 3 (March), 86–87.

LEWIS, J. P. 2001. Limits to software estimation. *ACM SIGSOFT Software Engineering Notes 26*, 4 (July), 54–59.

MALENFANT, D., 2000. Postmortem: Wild 9. Gamasutra - The Art & Business of Making Games, January. Available online at http://www.gamasutra.com/features/20000107/wild9_01.htm.

MEYNINK, T., 2000. Postmortem: Resident evil 2 for the n64. Gamasutra - The Art & Business of Making Games, July. Available online at http://www.gamasutra.com/features/200000728/meynink_01.htm.

MOLYNEUX, P., 2001. Postmortem: Black & white. Gamasutra - The Art & Business of Making Games, June. Available online at http://www.gamasutra.com/features/20010613/molyneux_01.htm.

MYLLYAHO, M., SALO, O., KÄÄRIÄINEN, J., HYYSALO, J., AND KOSKELA, J. 2004. A review of small and large postmortem analysis methods. In *IEEE France*, 17th International Conference Software & Systems Engineering and their Applications.

NATO. 1968. Software engineering - report on a conference sponsored by the nato science commitee. Tech. rep., Nato Science Commitee, Garmisch, Germany, October.

PRESSMAN, R. S. 2006. *Engenharia de Software*, 6ª ed. McGraw-Hill, São Paulo.

PRITCHARD, M., 2000. Postmortem: Age of empires ii: The age of kings. Gamasutra - The Art & Business of Making Games, March. Available online at http://www.gamasutra.com/features/20000307/pritchard_01.htm.

RAGAINI, T., 2002. Postmortem: Asheron's call. Gamasutra - The Art & Business of Making Games, May. Available online at http://www.gamasutra.com/features/20000525/ragaini_01.htm.

REINHART, B., 2000. Postmortem: Unreal tournament. Gamasutra - The Art & Business of Making Games, June. Available online at http://www.gamasutra.com/features/20000609/reinhart_01.htm.

RIDGWAY, W., 2000. Postmortem: Rangers lead the way. Gamasutra - The Art & Business of Making Games, February. Available online at http://www.gamasutra.com/features/20000201/ridgeway_01.htm.

SALADINO, M., 1999. Postmortem: Star trek: Hidden evil. Gamasutra - The Art & Business of Making Games, November. Available online at http://www.gamasutra.com/features/19991119/startrekpostmortem_01.htm.

SCHAEFER, E., 2000. Postmortem: Diablo ii. Gamasutra - The Art & Business of Making Games, October. Available online at http://www.gamasutra.com/features/20001025/schaefer_01.htm.

SCHEIB, V., 2001. Postmortem: Beam runner hyper cross. Gamasutra - The Art & Business of Making Games, November. Available online at http://www.gamasutra.com/features/20011109/whoopas_01.htm.

SMITH, B., 2001. Postmortem: Tropico. Gamasutra - The Art & Business of Making Games, october. Available online at http://www.gamasutra.com/features/20011010/smith_01.htm.

SOMMERVILLE, I. 2001. *Software Engineering*, 6th ed. International computer science seres. Addison-Wesley, London.

SPANEL, O., AND SPANEL, M., 2001. Postmortem: Operation flashpoint. Gamasutra - The Art & Business of Making Games, December. Available online at http://www.gamasutra.com/features/20011219/spanel_01.htm.

STANDISH GROUP. 1995. Chaos. Tech. rep., Standish Group.

STOJSAVLJEVIC, R., 2000. Postmortem: Westwood studios' command and conquer: Tiberian sun. Gamasutra - The Art & Business of Making Games, April. Available online at http://www.gamasutra.com/features/20000404/tiberiansun_01.htm.

UPTON, B., 2000. Postmortem: Rainbow six. Gamasutra - The Art & Business of Making Games, January. Available online at http://www.gamasutra.com/features/200000121/upton_01.htm.

VANDENBERGHE, J., 1999. Postmortem: The x-files. Gamasutra - The Art & Business of Making Games, December. Available online at http://www.gamasutra.com/features/19991203/xfiles_postmortem_01.htm.

WIKIPEDIA, 2007. Crunch time (expression). Wikipedia, The Free Encyclopedia, July. Available online at http://en.wikipedia.org/wiki/Crunch_time_(expression).

YOURDON, E. 2003. *Death March*, 2ª ed. Prentice Hall PTR, Estados Unidos, December.

Game	Unreal or ambitious scope	Feature creep	Cutting features	Design problems	Delay or optimistic schedule	Technological problems	Crunch Time	Lack of Documentation	Communication problems	Tools problems	Test Problems	Team building problems	Great number of defects	Loss of Professionals	Over Budget	Total	%
Beam Runner Hyper Cross	No	Yes	Yes	Yes	No	No	Yes	Yes	No	No	No	No	Yes	No	Yes	7	46,7%
Gabriel Knights	Yes	Yes	Yes	No	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	13	86,7%
Black & White	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	Yes	No	No	Yes	No	No	9	60,0%
Rangers Lead the Way	Yes	Yes	Yes	Yes	Yes	No	No	No	No	No	No	Yes	No	Yes	No	7	46,7%
Wild 9	Yes	Yes	Yes	No	No	Yes	No	Yes	Yes	No	No	Yes	No	Yes	No	8	53,3%
Trade Empires	No	Yes	Yes	Yes	No	No	No	No	No	No	Yes	No	No	No	No	4	26,7%
Rainbow Six	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	No	No	Yes	Yes	Yes	Yes	No	11	73,3%
The X-Files	Yes	No	No	Yes	Yes	Yes	Yes	No	No	Yes	No	No	No	No	No	6	40,0%
Draconus	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	No	Yes	No	Yes	12	80,0%
Cel Damage	No	Yes	Yes	Yes	No	No	No	No	No	No	No	No	No	No	No	3	20,0%
Comand and Conquer: Tiberian Sun	Yes	Yes	Yes	No	Yes	Yes	No	No	No	No	No	No	No	No	No	5	33,3%
Asheron's Call	Yes	Yes	Yes	No	Yes	No	No	Yes	No	No	Yes	Yes	No	No	No	7	46,7%
Age of Empires II: The Age of Kings	Yes	No	No	No	Yes	Yes	Yes	No	Yes	No	No	No	No	No	No	5	33,3%
Diablo II	Yes	No	No	No	Yes	Yes	Yes	No	No	Yes	No	No	No	No	No	5	33,3%
Operation Flashpoint	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	No	Yes	Yes	12	80,0%
Hidden Evil	Yes	No	Yes	Yes	No	No	No	No	Yes	Yes	No	No	No	No	No	5	33,3%
Resident Evil 2	Yes	No	No	Yes	Yes	Yes	No	No	No	Yes	Yes	No	No	No	No	6	40,0%
Vampire: The Masquerade	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No	No	No	No	Yes	No	No	7	46,7%
Unreal Tournament	No	Yes	No	Yes	No	No	No	Yes	Yes	Yes	No	Yes	No	No	Yes	7	46,7%
Tropico	No	Yes	Yes	Yes	No	No	No	Yes	No	No	No	No	No	No	No	4	26,7%
Occurrences	15	15	14	13	13	12	9	8	7	7	7	7	6	5	5	**143**	**7,2**
%	75%	75%	70%	65%	65%	60%	45%	40%	35%	35%	35%	35%	30%	25%	25%	**47,7%**	**47,7%**

Table 2: *Occurrence of problems in the projetcs*

RTSAI: a game tool for AI research

André M. C. Campos
Computer Science and Applied Mathematics Dept. - DIMAp
Federal University of RN - UFRN, Brazil

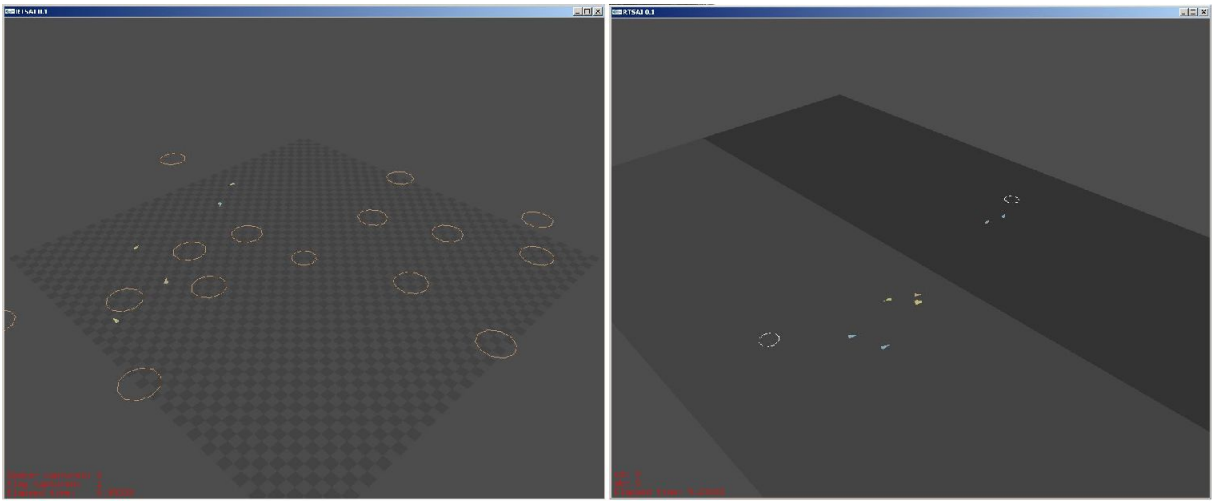


Figure 1: *Screenshots of applications using RTSAI. They are versions of the Capture-the-flag game.*

Abstract

In order to explore and evaluate new AI strategies, several researches have been conducted on gaming simulators, specially those dealing with realtime (e.g. RTS games). However, adapting a simulator for AI experiments usually requires great efforts. Aiming to facilitate this procedure, this paper presents a tool for developing new agent-based strategies, constructed on a real time game perspective.

Keywords:: Real-time Strategy Game, Multiagent system, AI engine

Author's Contact:

andre@dimap.ufrn.br

1 Introduction

Real-Time Strategy Game (also known as RTS game) is among the game genres which provide more challenges for AI researchers and it is, as consequence, also deeply influenced by AI results. It is actually intrinsically related to development of AI methods and algorithms that process events in a real-time fashion. This class of game illustrates the need of mechanisms that quickly respond for events, from the end-user, the network or the simulation kernel. M. Buro presented a detailed analysis and correlation between RTS games and AI research development, as the former deal with problems like real-time planning, decision making under uncertainty, machine learning, spatial and temporal reasoning, resource management, collaboration and navigation over the terrain, among others [Buro and Furtak 2004].

In order to explore and evaluate strategies for the aforementioned problems, it is usually necessary to develop the whole environment and the underlined mechanisms behind it. Since this option requires great efforts to put in place, several researches have been conducted on RTS gaming simulators, as an attempt to speed up the modeling of the problem being addressed. However, there is no universal game simulator which can address any type of problem. Furthermore, some existing tools do not give support for AI. They are harder to adapt to AI problems then others.

The objective of this paper is to present a new tool for facilitating the development of new AI strategies 1) in a game-based environment and; 2) for games or game-like applications. It addresses a specific class of problems, aiming to provide a flexible way to design and implement cooperative mechanisms dealing with issues in real-time. The tool, named RTSAI - *Real-Time Strategy for Artificial Intelligence research*, is based on the multiagent approach.

Besides its uses in games, RTSAI can be used as the logical layer of other application types, like ecosystem simulations (some ideas were extracted from the framework MAVIS [Campos and Hill 1998]). As a research tool, RTSAI can also be used for teaching AI techniques, where the students can learn by trying their own algorithms. Then, more specifically, the aim of this work is to provide a tool where students and researchers can develop and evaluate theirs algorithms and mechanisms in a game-based environment and following a multiagent approach.

The next sections detail the RTSAI, starting by a brief presentation of related tools. The section is required to distinguish RTSAI from other similar tools. In the following section, the RTSAI is detailed and its main characteristics are exposed. Next, the use of the tool in a multiagent course is presented. Finally, the paper is concluded by exposing the current state and next steps.

2 Related Work

There are several game tools to help the development of the behavioral layer of a game. It is possible to enumerate from games, where unit characteristics and behaviors are configurable by the end-user, to AI tools which can be integrated into gaming simulators. In this panorama, we have distinguished two main classes of tools related to RTSAI.

The first one is related to Open-Source games where it is possible to modify the unit behaviors by including our own code. There are several examples from this class. We can illustrate some of them by citing the games Glest [Figueroa et al. 2003] and Globulation2 [Staff 2004]. All of the mentioned games are RTS games, and one can evaluate new strategies by modifying the unit behavior code, usually written in script languages.

Globulation2 has an interesting feature particularly required for AI research. Differently from other RTS games, it aims to minimize the micro-management of the units. The player assigns high level tasks to the units and they try to achieve them with AI techniques.

This game provides them a rich environment where it is possible to investigate, for instance, multiagent planning techniques in order to transform high level users orders into low level units individual actions.

The main advantage of those applications is that the virtual environment where the research is conducted are already defined. There is no need to configure the problems since that are already there. The main disadvantage, in opposite, is the lack of flexibility for defining new research scenarios. Another great disadvantage is the effort required to set up our own algorithms, which may be hard coded into the game.

The second class are RTS game engines. Although most of them do not explicitly provide tools for AI researchers, they provide a common framework where it is possible to easily configure a research scenario. A well-known example is the Stratagus engine [Ponsen et al. 2005], which started from the FreeCraft project (a Warcraft II clone). Despite the fact that Stratagus is being widely used on AI research, there are some other engines more focused on this issue. Some of them are ORTS [Buro and Furtak 2005] and JaRTS [Filho et al. 2006]. Both engines aim to explore AI problems in a RTS game-based scenario. The focus is then to explore AI techniques, without worring about other components of the game, like the simulation kernel or graphics output.

ORTS was designed as a base tool for being used in challenge competitions in order to promote investigation on RTS-related problems. The idea is similar to what happens in the RoboCup challenge [Kitano et al. 1997]. The second example, JaRTS, was developed to help the investigation of new AI strategies in RTS games. JaRTS is a gaming simulator that also brings the idea of strategic competition behind it. Some of its designed choices was inspired from the Robocode environment [Nelson 2005].

These tools are targeted to RTS games, which means that most of the problems found on this game genre (resource allocation, decision making under uncertainty , etc) are well addressed. The RTSAI share with these tools the approach of using it as a competitive environment. Users must develop their own strategies and put them against others (also from other users) in order to evaluate the best one. The biggest difference between then and RTSAI is however that the latter aims to achieve more flexibility by not restraining to only RTS game scenarios. RTSAI can be used as the logical layer of different type of application, from RTS games to real time simulations. The next section details how such a difference is achieved by presenting the RTSAI project.

3 RTSAI project

RTSAI belongs to the second class (mentioned in the previous section). It was designed to facilitate the investigation of new AI strategies for RTS games. However, it is not restraint to this application. It was designed from a multiagent perspective. So, the behavior modeling task focuses on specifying behaviors of game units individually. Each of them is considered as an autonomous entity (also known as agents) and may react to changes on the game scenario (also known as environment events). In RTSAI, such a modeling is made through the use of script files, which aim to facilitate the iterative and cyclic process of: 1) behavioral modeling, 2) implementation, and 3) verification and validation.

3.1 Design choices

RTSAI focused on the configuration of a gaming scenario and the behavioral model that is going to run over it. The behavior model uses AI strategies in a low level of abstraction. The latter tries to give support to the development of high level task, like the ones found in Globulation2. Those three main aspects – abstraction level, scenario (environment), and behavior model – are described hereafter.

3.1.1 AI abstraction level

One can classify AI methods into two or more level of abstractions. The lowest ones deals with basic operations, for instance, the intelligent displacement of a unit. In order to tackle this issue, there are some algorithms like A* or methods like steering behaviors [Reynolds 1999]. Those basic mechanisms can be used for solving more complex tasks, considered to be in a higher level of abstraction. Some examples are the patrolling [Menezes et al. 2006] and the discovering [Soares and Campos 2006] problems. The latter can, on their way, be used to solve problems in a higher level of abstraction, and so on.

There exists then a dilemma when designing a AI supporting tool: which level of abstraction the tool is going to tackle. The dilemma resides on the fact that the more abstract (high level) the less flexible it is. In other words, when a tool is target to high level tasks, it usually works well for a limited set of tasks. On opposite, when a tool is target to low level requests, the tasks can be composed by the user to achieve several types of objectives, being in this way more flexible.

Being a tool for general use (including non-game applications), its focus is turned to basic mechanisms in a low level of abstraction. Some of the implemented mechanisms are the following.

- Search algorithms (A*);
- Steering behaviors;
- Basic communication services (individual, group and broadcast messaging);
- Basic coordination mechanisms (organizational structure, interaction protocols).

At the moment, the user of the tool must reuse and combine them to compose solutions for more complex problems. An on going research is being conducted to provide a planning mechanism able to generalize a set of high level problems. This mechanism is an adaptation of DHSM - Dynamic Hierarchical State Machine [Wagner 2006]. The adaptation is based on a contextual approach, where a specific state can be reused on different contexts (situations). The state machine, implementing a unit behavior, is then dynamically created to achieve the unit objectives by reusing already defined states on the current context and it is also considered a new state, which might also be reused further on.

3.1.2 Environment

Another important design choice when defining an agent-based tool is to specify how to model the environment. There exists several forms to structure the way in which spatial informations will be gathered (regular grids, BSP, navigation graphs, among others). The choice of a unique option would certainly restraint the techniques involved in a particular application. For instance, some techniques (e.g. steering behaviors) are more suitable for continuous-based environment, while others (e.g. A*) suit well in discrete-based environments (e.g. using regular grids). In order to be more general, RTSAI is being structured to represent the environment through several levels of abstraction. In other words, the modeled environment, can be composed of several representation layers hierarchically separated.

The first layer represents the environment in a continuous way, i.e. the environment is defined by the elements over it and their respective positions (in real values). On this level, there is no environment structure. In order to a unit inquire about its surrounding units, it is necessary to cover all existing units. Then, this layer is mostly used only for small spatial coordination problems, involving few units and/or objects. The second layer organizes the environment through a regular rectangular grid and is well suited for larger environments taking into account a larger number of units. Only the neighborhood grid cells are consulted to inquire about the surrounding units or objects. Finally, the third and last layer structure the environment through a navigation graph. In this layer, every graph node is associated to a grid cell (established on the previous layer), making possible to implement hierarchical path planning [Botea et al. 2004].

Up to the moment, there is no supporting tool to help to set up the environment (for instance, a level design tool). All the information

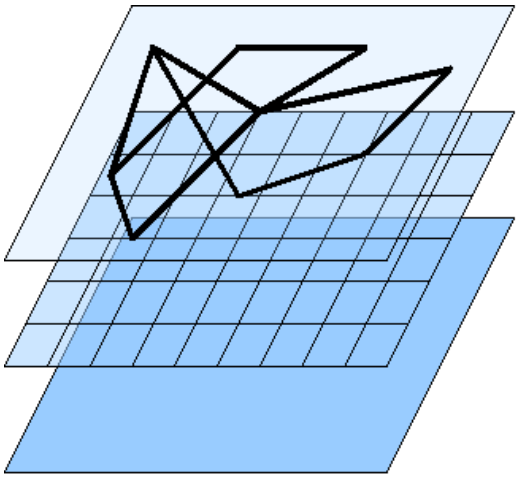


Figure 2: *Hierarchical layered structure of the environment. The first one models model the environment in a continous way. The second one splits the environment in a regular grid. The latter structures the environment through a navigation graph.*

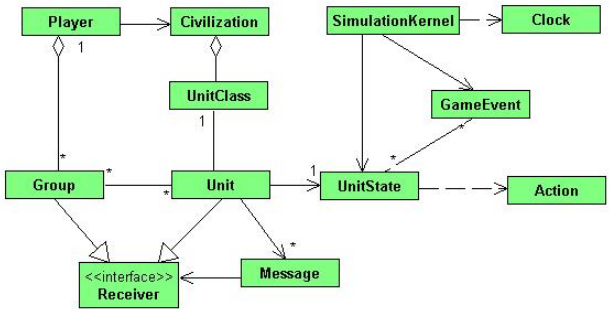


Figure 3: *RTSAI main classes*

must be set by hand in the related script (explained latter). The Figure 2 illustrates this layered structure showing up the higher level layer as a navigation graph, the intermediary layer as a rectangular grid, and the lowest level as a continuous environment.

3.1.3 Behavior

RTS games involve the coordination of actions generated by several units. RTSAI follows an agent-oriented approach to model this team behavior, i.e. a behavior is individually assigned to each unit and it acts as an autonomous entity. In order to model the individual behaviors, an hierarchical state machine, similar to the machine proposed by [Kessler et al. 2004], was used. A clear difference is the ability to reuse the state in different contexts. In order to facilitate behavior prototyping, as well as to configure the set of possibles events in the environment and the actions that the unit can perform, all of them are configurable by script.

3.2 Project design

The RTSAI main classes are presented in Figure 3. When a player is created in the game (instance of `Player`), it is assigned the civilization which s/he belongs to (`Civilization`). A civilization defines a set of unit classes (`UnitClass`), specifying particular attributes like velocity, strength, etc. The unit (`Unit`) created by a player must belongs to a unit class defined on the player civilization. The units are grouped (`Group`) on at least one group, the player group, which has all player units. However, other groups can be formed during the game play. The group is useful for coordinating agents. They can exchanges messages (`Message`) between members of the group, as well as individually (peer-to-peer). The behavior of a unit is defined by the state it is currently on (`UnitState`). A unit follows the behavior written on its current state by performing actions (`Action`). The latter might be shared by several agents. On each gaming simulation loop, the simulation kernel (`SimulationKernel`) checks the occurrence of the registered events (`GameEvent`), activating the units which states are listening to them.

Some other utilitarian classes are also part of the RTSAI architecture, like classes for math and unit pooling (in order to optimize unit creation and destruction). Those classes are omitted here for clarity. RTSAI was designed to be used together with a third-part visualization components, either 2D or 3D. However, for easily debugging behavior without worrying about integration tools, a tiny graphical layer was introduced in RTSAI. The implementation was made in C++ using Lua [Ierusalimschy et al. 2005] as scripting language.

The Lua scripts are responsible for setting up the whole game environment (the number of players, their civilizations, their initial units, which classes they belongs to and initial states) as well as the behavior strategy of each player. A player strategy is a set of unit states instances which, together, defines a hierarchical state machine. The dynamic aspect of RTSAI is better explained in the next section.

4 Using the RTSAI

The configuration of an application using RTSAI is based on the implementation of three types of scripts (based on Lua language). They define:

- **The environment**: This file specifies the existing game events, the actions an unit may perform, the type of objects on the environment and its initial state. Just one environment file is loaded.
- **The behavioral model**: Several behavioral files can be loaded, usually one per player. It specifies the player state machine. Each state must be set to answer what to do when an event is triggered.
- **The units classes**: This files specifies the set of unit classes. Each class has different values for different attributes, like speed, strength, resistance, etc.

4.1 Setting the environment

The file describing the environment models the game-related problem that the researcher is trying to solve (or the world being simulated). It contains the following five attributes:

1. The world: it models how the environment is structured (e.g. 512x512 a rectangular grid);
2. The players: it specifies which players are involved and their respective strategies;
3. The events: it defines the set of events modeled in the world, specifying in which conditions they happen;
4. The actions: it is the set of actions the units might perform, specifying how the environment is changed by them;
5. The world initial state: it is the world initial configuration (where the units are positioned, etc).

The Figure 4 illustrates the definition of an environment on RTSAI. The `players` attribute assigns a script file to each player. This file is the player behavioral model (described latter). It also assigns a player to a specific civilization (set of unit classes), where the unit attributes are defined. The `events` attribute is used to register which events are able to be listen by the units. On each game simulation iteration, the registered events are checked (by their function on the script). Whenever a function returns an information (a non-empty lua table), it means that the event happened. The returned information is then passed through for all units listening to such an event. The `action` attribute determines the set of actions that an agent might perform. The action specification is defined by two functions. While the first one is used to check if the environment

fits the action preconditions, the second one is the script for executing the action, i.e. it modifies the environment. The other attributes – `world` and `objects` are responsible for modeling the environment and the type of resources (objects) over it. Finally, the function `init`is responsible for initializing the environment, creating the resources, the units for each player and so on.

```
Game {
  players = {
    p1 = {
      civilization = "civilization_file",
      strategy = "a_behavior_file"
    },
    p2 = {
      civilization = "civilization_file",
      strategy = "another_behavior_file"
    }
  },

  world = { width = 512, height = 512, grid_size = 8 },

  events = {
    an_event = function ()   -- creates "an_event"
      -- check the event and return its info
    end
  },

  actions = {
    an_action = {  -- creates the action "an_action"
      precondition = function( self, unit )
        -- check preconditions (e.g. is it blocked?)
      end,
      execute = function( self, agent, elapsed_time )
        -- change the environment (e.g. move the unit)
      end
    }
  },

  objects = {
    -- resources in the environment
    -- e.g.: goldmine = circ (5, true)
  }

  init = function ( self )
    -- initializes the game scenario
  end
}
```

Figure 4: *Environment setting example.*

4.2 Setting the behavioral model

One of the previously described attribute is the players attribute, which must specify the behavior file used by each player. This file describes the several states that the units belonging to that player might be set. Each state may specify the operation the units will perform when entering in the state (`on_enter`), when leaving it (`on_exit`), as well as when on it (`act`). The `act` function should return one of the actions registered in the environment (previously mentioned when describing the `actions` attribute) or an already predefined action on RTSAI (for instance, `move`).

When defining a state, one can determine how the units will reply when a message is received. This is done by implementing the function `on_message`. The unit may then engage in a conversation by replying the received messages. In order to structure unit conversations, RTSAI uses a subset of message performatives proposed by FIPA ACL [FIPA 2002]. The message transfer is considered an event, however it is just captured by the receiver unit or group (when a message is broadcast).

It is also possible to specify behaviors to other type of events. The Figure 5 illustrates the code of a state named `a_state`. In this example, the state registers that all units in this state listen to the event named `an_event` (defined on the environment file). It is done by implementing a callback for the event (`on_an_event` function). The existence of the described features in the state machine provide the ability to mix the Moore and Mealy state machine models [Wagner 2006].

```
State {
  -- STATE ATTRIBUTES ---------------------------------
  name = "a_state",
  a_state_attribute = 10,

  -- STATE FUNCTIONS ----------------------------------
  on_enter = function ( self )
    -- execute when entering in the state
  end,

  on_exit = function ( self )
    -- execute when leaving
  end,

  act = function ( self )
    -- return an action for each game loop iteration
    action.move.steer = self.unit:steerForSeek(a_target)
    return action.move
  end,

  -- EVENT CALLBACKS ----------------------------------
  on_an_event = function ( self, event )
    -- answering to "an event".
    -- e.g. change to "another_state"
    return another_state:new()
  end,

  on_message = function ( self, messages )
    -- process receiving messages
  end
}
```

Figure 5: *State machine example.*

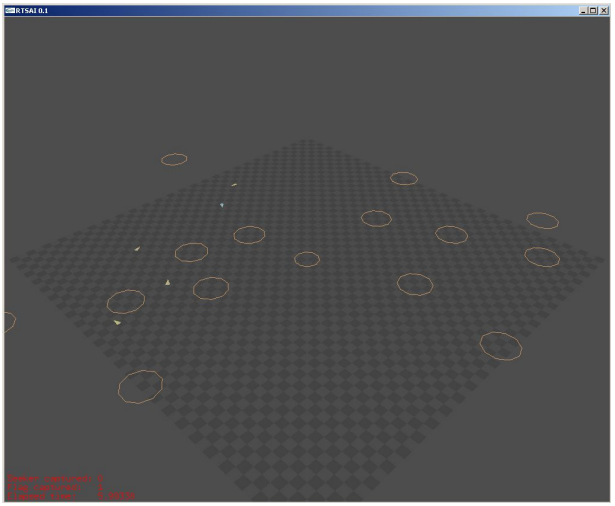


Figure 6: *The simpler version of the Capture-the-flag game.*

5 Application example

As previously mentioned, RTSAI gives support for implementing the logical layer of real time games and simulations. This is done by facilitating the units behavior modeling through an dynamic hierarchical state machine. Several applications can be constructed on top of it.

Up to date, two application prototypes were developed. They were developed to be used as a AI learning tool in the “multiagent systems course” offered for undergraduate and graduated students of the Federal University of RN (2007.1 semester). The use of a game-based tool for teaching provided a funny environment where the students could experiment their own AI strategies for solving complex problems. The challenge was to develop the better strategy, competing with the colleagues. Such an approach was very well accepted by the students, who were (all of them) strongly motivated by the practical activity. The experience also showed the relevance of using game-based environments as a learning technique.

Both prototypes model the classical *Capture-the-flag* game (see Figures 6 and 7). However, the first one required the modeling of the behavior of only a unit. This is a simplified version of the

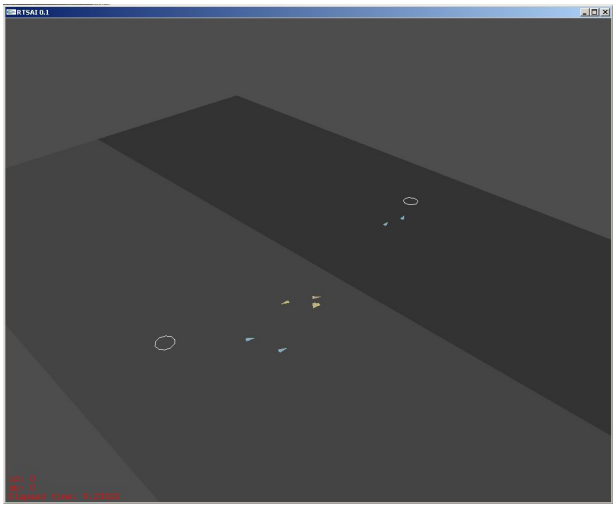


Figure 7: *The full version of the Capture-the-flag game (player vs. player).*

game, where there is just an unit trying to avoid several enemies while attempting to catch the flag in the center of the environment. The student must to implement the unit state machine using strategies like individual planning and reactive behaviors (steering behaviors). The winner is who catch the flag more times in a predefined time.

The second prototype models a full version of the *Capture-the-flag* game. In this game, two players play against each other to capture the enemy flag (as much as they can). In this version, the student are faced to more complex issues since there are four units to control and coordinate. While the problems found in the previous prototype still exists, however the student needs now to deal with multiagent planning and coordination. The students could also visualize the emergent property of multiagent systems.

Although the prototypes are not RTS games, since they do not deal with unit classes, resource gathering etc., they consist of a basis for strategic games requiring real time feedback. In order to implement the visualization component of the prototypes, a graphic layer was adapted from the OpenSteerDemo, used to demostrated the OpenSteer library [Reynolds 2003]. They were then developed using the OpenGL and Glut graphical libraries.

6 Final remarks

This paper presented a tool which aims to give support to the development of strategic game-based applications involving real time. It is then well suited for investigating AI techniques on problems related to RTS games. This tool, named RTSAI – Real-Time Strategy for Artificial Intelligence research, was currently used for configuring scenarios in game-like applications in order to used them on an academic course. Its use showed that the tool is flexible enough and that it provides a useful set of features to help game developers and researchers.

Despite the objective of facilitating the development of game-like applications, the flexibility promoted by RTSAI compromise, however, its facility of use. This is due to the fact that, for testing strategies for new problems, it is necessary to implement the problem scenario (game context). That was a design choice. If a tool targets a specific application domain, it would certainly be more easy to set up problems for such a domain. However, it would not be useful for many other type of applications. RTSAI tries to find a compromise between generality and specificity in order to give a useful set of features to RTS-like games and applications.

Although it is already operational, RTSAI is in continuous development. Some limitations are being addressed for future versions. The planning mechanism, previously mentioned, is one of them. Other applications are also being developed on top of it. A complete RTS game is under construction.

References

BOTEA, A., MÜLLER, M., AND SCHAEFFER, J. 2004. Near optimal hierarchical path-finding. *Journal of Game Development 1*, 1.

BURO, M., AND FURTAK, T. 2004. RTS games and real-time AI research. In *Proceedings of the 13th Conference on Behavior Representation in Modeling and Simulation (BRIMS)*.

BURO, M., AND FURTAK, T. 2005. On the development of a free RTS game engine. In *Proceedings of GameOn'NA Conference*.

CAMPOS, A., AND HILL, D. 1998. An agent based framework for visual-interactive ecosystem simulations. *SCS Transactions on Simulation 15*, 4, 139–152.

FIGUEROA, M., GONZÁLES, J., FERNÁNDEZ, T., AND ZANNI, J., 2003. Glest: A Free 3D Real-Time Strategy Game. `http://www.glest.org/` [accessed on 27/02/2007].

FILHO, V., WEBER, R., JÚNIOR, J., RAMALHO, G., AND TEDESCO, P. 2006. Jarts: Java RTS simulator. In *Digital Proceedings of the V Brazilian Symposium on Computer Games and Digital Entertainment*.

FIPA, 2002. FIPA ACL message structure specification. `http://www.fipa.org/specs/fipa00061/index.html` [accessed on 27/02/2007].

IERUSALIMSCHY, R., DE FIGUEIREDO, L. H., AND CELES, W. 2005. The implementation of lua 5.0. *Journal of Universal Computer Science 11*, 7, 1159–1176.

KESSLER, R., GRISS, M., REMICK, B., AND DELUCCHI, R. 2004. A hierarchical state machine using JADE behaviours with animation visualization. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'04)*.

KITANO, H., ASADA, M., KUNIYOSHI, Y., NODA, I., AND OSAWA, E. 1997. Robocup: The robot world cup initiative. In *Proceedings of the first international conference on Autonomous agents*, ACM Press, New York, NY, USA, 340–347.

MENEZES, T., RAMALHO, G., AND TEDESCO, P. 2006. Negotiator agents for the patrolling task. In *Proceedings of Brazilian Symposium on Artificial Intelligence (SBIA'06)*.

NELSON, M., 2005. Robocode: An Open Source Educational Game. `http://robocode.sourceforge.net/` [accessed on 27/02/2007].

PONSEN, M., LEE-URBAN, S., MUÑOZ-AVILA, H., AHA, D., AND MOLINEAUX, M. 2005. Stratagus: An open-source game engine for research in real-time strategy games. In *Workshop for International Joint Conference on Artificial Intelligence (IJCAI-05)*.

REYNOLDS, C. 1999. Steering behaviors for autonomous characters. In *Proceedings of the Game Developers Conference*, 763–782.

REYNOLDS, C., 2003. OpenSteer: Steering Behaviors for Autonomous Characters. `http://opensteer.sourceforge.net/` [accessed on 27/02/2007].

SOARES, G., AND CAMPOS, A. 2006. Multiagent exploration task in games through negotiation. In *Digital Proceedings of the V Brazilian Symposium on Computer Games and Digital Entertainment*.

STAFF, L. J. 2004. upfront. *Linux J. 2004*, 127, 18.

WAGNER, F. 2006. *Modeling Software with Finite State Machines: A Practical Approach*. Auerbach Publications.

RTSCup Project: Challenges and Benchmarks

Vicente V. Filho Clauirton A. Siebra José C. Moura Renan T. Weber Geber L. Ramalho

Universidade Federal de Pernambuco, Centro de Informática, Brazil

Abstract

Real Time Strategic (RTS) games are an interesting kind of domain to Artificial Intelligence (AI) community due to the problems and challenges that such games can offer. In fact, several AI techniques can be applied to solve RTS problems, such as team coordination, pathfinding or patrolling. In this context, this paper presents the RTSCup Project a new RTS simulator, which can mainly be used as benchmark for multiagent systems. The architecture and main concepts of this project are discussed in details, as well as the advantages of its use.

Keywords: artificial intelligence, simulator, digital entertainment, benchmarks.

Authors' contact:
{vvf,cas,jcmj,rwt,glr}@cin.ufpe.br

1. Introduction

The evaluation of any computational system is an important phase of its development process and this is not different to systems that use Artificial Intelligence (AI). For such systems, in particular, there is a trend [Hanks et al. 1993] in employing simulation environments as benchmarks, which enable the evaluation of such systems in different test scenarios. Another interesting trend is the use of such environments in academic competitions [Stone 2003]. This trend brings two advantages to AI research. First, competitions motivate the investigation of solutions in specific areas. Second, they integrate the research community, providing an environment where approaches are discussed and results, raised from contests, can be analyzed in a comparative way.

The aim of this work is to present a simulator to real-time strategic games, called RTSCup, which can be used as a benchmark for evaluating several AI techniques used to implement teams as Multiagent Systems (MAS). RTSCup was already employed in a practical experiment during a competition among AI students of the Federal University of Pernambuco, as discussed along this paper.

The remainder of this work is structured in three major parts: presentation of the RTS game as a multi-agent environment; identification of the RTS game as benchmark in the multi-agent area, and presentation of the RTSCup simulator project and architecture.

2. RTS game as a MAS

A RTS game is a specific but very attractive real-time multi-agent environment from the point of view of distributed artificial intelligence and multi-agent research. If we consider a RTS team as a multi-agent system, several interesting issues will arise.

In a game we can have two or more competing teams. Each team has a team-wide common goal, namely to win the game. The opponent team can be seen as a dynamic and obstructive environment, which might disturb the achievement of this goal. To fulfill the common goal, each team is required to complete some tasks, which can be seen as sub-goals (i.e., foraging, exploration, fight and resource allocation). To achieve these sub-goals each team must present a fast, flexible and cooperative behavior, considering local and global situations.

The team might have some sort of global (team-wide) strategies to accomplish the common goal, and both local and global tactics to achieve sub-goals. Furthermore, the team must consider the following challenges:

- The game environment, i.e. the movement of the team members and the opponent team is highly dynamic.
- The perception of each player could be locally limited (i.e., fog of war).
- The role of each player can be different (i.e. worker collects resources and tanks attacks).

In brief, based on these issues a RTS team can be seen as a cooperative distributed real-time planning scheme, embedded in a highly dynamic environment. Besides, a RTS game can be viewed as a MAS considering the following main features: (1) each agent has incomplete information or capabilities for solving the problem and, thus, it has a limited viewpoint; (2) there is no system global control; (3) data are decentralized; and (4) computation is asynchronous.

3. Benchmarks and MAS

The use of benchmarks, as evaluation method for multi-agent systems, has become a common practice for AI systems. In fact, this method is able to evaluate the performance of such systems and provide a basis for comparative analysis. However, some criticisms [Hanks 1993] have been directed against the use of benchmarks. First, such method is an empirical form of

evaluation: its users have to distinguish the important evaluation events, as well, interpret the results of such events. Second, there is not a consensus about what a representative benchmark is. Finally, results from benchmark experiments are not general. Rather, they are related to a subgroup of possibilities from the total set of scenarios.

Simulators are a particular type of benchmark, whose generation of new states is executed in runtime and such states depend on the activities performed inside the environment. In this way, final results are commonly unpredictable. An interesting trend related to simulators is academic competitions [Stone 2003]. Besides, motivating research and integrating the scientific community, competitions determine deadlines to the creation of functional systems and periodic events, using the same platform, enables the improvement of past systems and their approaches.

One of the main competitions related to multi-agent systems is the *RoboCup Rescue* (RCR) [Kitaro and Tadokoro 2001]. This competition uses a real-time distributed simulation system composed of several modules, all of them connected via a central kernel. The function of this system is to simulate the effects of earthquakes in urban areas. For that purpose, each type of event related to an earthquake, such as building collapse or fire spreading, is simulated by a dedicated module, while a *Geographic Information System* (GIS) provides the initial conditions of the disaster space.

RCR is an appropriate example of benchmark to multi-agent research because it implements several of the requisites that such systems requires. Those are:

i. Agents do not have control on the environment, their actions are not the unique events that can change it;

ii. Agents are not able to ensure that a sequence of actions will lead to a specific state, or if these actions are valid because changes can happen over the environment between decision and its execution;

iii. RCR environments are complex and each of their objects presents several attributes whose values can affect the simulation progress;

iv. The environment considers communication and coordination among agents as an important simulation issue, there are specific rules to control such communication;

v. There are several ways to measure the efficiency of approaches, for instance, number of victims or total area of fire;

vi. The RCR simulator has a well defined temporal model, which is based on configurable cycles;

A last and important RCR feature is its high level of parameterization, which enables an evaluation of multi-agent systems considering a significant variety of problems and conditions. In this way, RCR users can configure the environment in such way that it can be more useful during evaluations of some particular aspect.

4. RTS as Benchmarks

As discussed in the last section, the use of benchmarks, as an alternative, to evaluate MAS has received several criticisms, which are mainly based on the fact that such systems are implemented to be used in real situations. In this way, independently of the specification level of a benchmark, it will still represent a limited number of situations that could happen in real scenarios.

There exist cases, however, in which realism is not the main requirement to be considered. In such situations, the goal could be focused on the comparative evaluation among different approaches. For example, benchmarks currently used in the *Planning Systems Competition* [McDermott 2000] and the *Trading Agent Competition* [Wellaman et al. 2001] corroborate this idea.

Another kind of competition, which is recently receiving more attention from the research community, is related to Real-Time Strategy (RTS) games. Note that the main competition in this area (*ORTS RTS Game AI Competition* [Buro 2003]) does not have the same maturity than other AI competitions. However, several benefits can already be observed, such as the creation of a community with common interests in RTS problems and their investigation.

One of the main advantages of using RTS environments as benchmarks is the broad variety of problem types that they can generate. For example, consider a classic RTS game of battle between two teams. Some of the problems that can be investigated in this case are:

i. Pathfinding: teams need to move along the best routes so that they decrease time and effort. It is common more elaborated versions of this problem, such as involving routes along unknown lands or facing dynamic obstacles (e.g., enemy teams);

ii. Patrolling: a team can keep a specific area on control, or cover an area to find resources or enemies in an optimized way;

iii. Resource allocation (schedule): each component of a team is a resource that must be allocated in some activity, such as defense or attack. Allocation processes must observe, for example, the *load balancing* of activities, specific

resources do not become overloaded while others are free;

iv. Actions prediction: a team can try to observe and model the behavior of others, it predicts their behaviors and can anticipate them;

v. Coordination: components of a team ideally need some kind of coordination to improve the whole work and they do not disrupt each other. For example, during an attack maneuver the team needs to decide if its members will attack the flanks, or if the infantry should wait by the intervention of the artillery before moving forward;

vi. Deliberative and reactive architectures: each team needs to deliberate on the strategies to be deployed in a battlefield. However, several reactive actions, such as reactions to eventual ambushes, must also be considered;

vii. Strategy and tactic decisions: each team must plan and conduct the battle campaign (strategy) in the same way that must organize and maneuver forces in battlefield to achieve victory (tactics).

Then, RTS environments enables a wide set of problems and situations, in which it's possible to apply AI techniques. Ideally, these environments must be configurable following the RCR model: users can create more appropriate scenarios to each kind of problem.

5. Main Simulators and Competitions

There are currently several simulators to RTS games and some of them are used in open competitions. In this section we discuss five examples: ORTS, Stratagus, Boson,Glest and JaRTS.

Open Real-Time Strategy (ORTS) [Buro 2003] is a simulation environment for studying real-time AI problems such as pathfinding, reasoning with imperfect information, scheduling and planning in the domain of RTS games. Users of ORTS can define rules and features of a game via scripts that describe several types of units, buildings and interactions. The ORTS server is responsible for loading and executing such scripts, sending the world state to their clients. These clients are applications that generate actions to be performed over the simulation server, according to their objectives and the current state of the world.

The ORTS competition [Buro 2006] comes up in cooperation with the *Artificial Intelligence and Interactive Digital Entertainment Conference* (AIIDE). There are four different contests in this competition:

i. Game 1: agents must develop strategies to perform resource gathering. The goal is to gather the maximum amount of resources within 10 minutes;

ii. Game 2: tank agents must destroy as many opponent buildings as possible within 15 minutes. However, at the same time, they must also defend their home bases;

iii. Game 3: apart the resources gathering and battle actions, agents must also deal with resource management, defining if the team should collect more resources or spend it. The decision making process involves, for example, actions to create more units or attack/defend specific positions. The main goal is to destroy all opponent buildings within 20 minutes.

iv. Game 4: marine agents must destroy as many opponent units as possible within 5 minutes.

Stratagus [Stratagus 2007] is an engine to RTS games that supports the development of both multiplayer online and single player offline games. Stratagus is configurable and can be used to create customized games. Some of these games made with Stratagus, such Wargus, are been used as a platform for AI studies.

Boson [Boson 2005] is a RTS engine to games similar to *Command & Conquer* and *StarCraft*. The battles take place in both land and air, with a high number of units fighting at the same time. An interesting feature of this simulator is the effect of world features (e.g., wind speed, natural obstacles such as trees, and terrain variety) on actions that are performed along the scenarios. Individual units have their own movement pattern, animation, multiple weapons and intelligent pathfinding, they can fight against other agents or human players.

Glest [Glest 2007] is a 3D RTS engine available for several platforms. The engine can be customized via XML files, so it's possible to define basic parameters such as life, magic, protection, and so on. The current version includes single player game against AI controlled players, providing two factions for player control: *Magic* and *Tech*, each of them with their corresponding tech trees, units and buildings.

Java Real-Time Strategy Simulator (JaRTS) [JaRTS 2006] is a RTS game simulator that enables users to focus their development uniquely on the agents' behavior. This means that users don't have to worry about the simulation and evolution of the environment. The approach of JaRTS is very similar to the *Robocode* simulator [Li 2002], where there are basic classes that implement the main behaviors of their agents, such as move to, twirl around, or shoot in. The task of programmers is to extend such classes to create more complex behaviors.

JaRTS presents three basic types of agents: (1) the W*orker* whose role is to collect resources to fill up the *Control Center*; (2) the *Tank* whose objective is to attack or defend specific positions around the environment, and; (3) the *Control Center* that represents a place to deposit collected resources and is also the target (to protect or to destroy) of *Tank* agents.

Each type of agent in JaRTS can be individually simulated. For example, we can create a *MyWorker* agent and simulate its behavior in an environment populated only with *Worker* agents. This means, every agent is collecting resources without interference of enemies so we can focus our attention on this specific task. Note that this is the scenario of ORTS Game 1. In the same way, it is possible to create a *MyTank* agent and simulate it in a world where the unique goal is to fight, without considering resources gathering or management, as happens in ORTS Game 2.

The implementation of an agent's behavior is executed in a similar way to the Robocode simulator. Users just extend one of the unit classes (*Worker*, *Tank* or *ControlCenter*) to specify a particular behavior. For example, users can create the class *MyWorker* extending the class *Worker*. At the simulation runtime, users can choose the instances of classes that will connect to the simulation environment

We can discuss some main points of this analysis. First, Boson and Glest simulators are not centered in AI issues, it would be necessary an additional effort to define and create test scenarios to evaluate AI techniques. Differently, Stratagus provides a script language (LUA) that allows researchers to modify the game AI without having to change the engine code. LUA employs many familiar programming paradigms from 'common' programming languages such as C (e.g., variables, statements, functions), but in a simpler fashion [Posen et al. 2005]. Stratagus was also integrated to TIELT - *Testbed for Integrating and Evaluating Learning Techniques* – [Aha and Molineaux 2004], which is a freely available tool that facilitates the integration of decision systems and simulators. Currently, the TIELT focus has been on supporting the particular integration of machine learning systems and complex gaming simulators.

ORTS has initially presented itself as a good candidate to be a MAS benchmark. However, after several trials, we have identified that it is complex to use, presenting a number of configuration problems and behavior during its performance. The ORTS' main problem is lack of proper documentation. It's hard to find out how to set it up, and how to execute it correctly (using your own solution) as well. Furthermore, there is not a methodology related to updates, which generates a new set of problems after each release causing "crashes" even on the simulator itself, not to mention the solution's ones.

JaRTS at the beginning was considered the best choice for AI students and researchers. It is very user friendly, with a GUI (Graphic User Interface) that helps to configure/run a simulation. Furthermore, the simulation concerns are all encapsulated in the server, so that users just have to implement the agent behavior by extending an abstract class and then load it to simulate. Despite these positive aspects, which are more applied to academic use as a "fun" way to study AI, the JaRTS approach is very simple to support a deeper AI research.

In this way, after an analysis of all our alternatives and problems, we have decided to create a new simulator that could be simpler and oriented to AI problems. Like a merge between ORTS' RTS complexity (different unit types, tech-tree, fog of war, etc.) and JaRTS' "user friendship" (encapsulated server, GUI, etc.). This means, users just need to be worried about solution related to AI problems. This simulator is presented in the next section. .

6. The RTSCup Project

The RTSCup Project was planned to maximize the community contribution and attain high-throughput in research itself. From the research perspectives, RTSCup was designed; (1) to enable users to focus their efforts on AI research; (2) to be used as benchmark in the multi-agent area; and (3) to integrate the research community.

6.1 Simulation Project

The RTS game simulation requires the simulation of different aspects of this game type. These aspects include motion simulation and collision avoidance; attack and cure; collect and deliver resources; build, train, and fix new objects (units and buildings); convert enemy units; research new technologies (upgrade civilization - also known as tech tree); and so on.

The RTSCup simulator is divided in the sub-systems presented below:

Motion and collision avoidance system: simulates the motion of all units in the game environment and is also responsible for collision detection. This system does not allow that fast moving objects can pass through each other without detection. A sweep test [Gomez 1999] is used to efficiently determine the exactly time of overlap.

Attack / Cure system: simulates occurrence of activities that modify the health points (hp) of objects in the game environment like an attack or a cure action. This system is responsible for the battle simulation in the simulator.

Collect / Deliver system: simulates the resource gathering activities. This system is responsible for the

consistency maintenance in the game world. A gold resource, for example, can be mined until its resources have finished. After that, the gold resource must disappear and the resource collected must be in some place in the map (e.g., units or buildings).

Build / Train / Fix system: simulates the creation of new units and buildings in the game world. Each building and unit normally has different roles in the game (i.e., the worker is used to gather resources and a tank to fight) and the user must create different types of units to reach the victory.

Religion / Convert system: in RTS games, all teams have their own religion with the respective units (e.g., monk or father) and buildings (e.g., church). These units are responsible for the convert action – change elements from one team to another one.

Research system: simulates the development of a civilization. Each team can have its own upgrade tree, also known as tech tree, which is an abstract hierarchical representation of the possible paths of research that a team can take. The tech tree is the representation of all possible paths of research a team can take. All leaves in the tech tree are defined by the following attributes: cost, pre-requisites, duration time and benefits. To research a new technology in the tech tree, it is necessary to fill some pre-requisites and pay a tax. After a pre-defined period of time, the research finishes and the team gains some benefit (e.g., increase attack range or decrease train costs).

6.2 Simulator Architecture

The simulation project involves the development of a comprehensive simulator that enables the integration of all above features in a multi-agent environment in a large scale. The simulator should be able to combine all above cited features and present them as a coherent scene. The current version of the simulator architecture is shown in Figure 1**.**

The RTSCup simulator is a real-time distributed simulation system that is built of several modules connected through a network. Each module can run on different computers as an independent program, so the computational load of the simulation can be distributed to several computers.

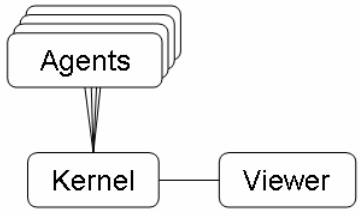


Figure 1: RTSCup Structure

The architecture is divided into the following modules.

- Kernel: combines all information and updates the game world. In the kernel is presented all the main systems mentioned in the above section.

- Agent: one of the RTSCup simulator modules that communicates with the kernel through a network to act in the game environment. Each type of agent is controlled by a program written by the user. This means, all user's workers could act the same way, since all of them run the same source code. The agents must be prepared to negotiate with other agents to define its tasks.

- Viewer: another RTSCup simulator module, as the name says, it is used to visualize the game environment status.

The RTSCup simulation occurs in two steps: Initialization and Progress. The Initialization is the first step and occurs only once. Differently, the Progress step is repeated until the end of the game.

Initialization Step: the kernel reads the configuration files to load the initial condition of the game world. Now the kernel is ready to receive connections from viewer and clients. RTSCup clients connect to the kernel indicating their agent type and team. Then the kernel assigns each unit and buildings in the virtual world to its corresponding behavior according to the type and team, sending the initial condition related to each agent's cognition.

Progress Step: when all unit and the buildings have been assigned to RTSCup agents, the kernel finishes the integration and the initialization of the kernel. Then, the simulation starts. The viewer has to be connected to the kernel before all the RTSCup agent assignments have been carried out.

The simulation proceeds by repeating the following cycle. At the first cycle of the simulation, steps 1 and 2 are skipped.

i. The kernel sends individual vision information to each RTSCup agent.
ii. Each RTSCup agent submits an action command to the kernel individually.
iii. The kernel sends action commands of RTSCup agents to specific sub-system.
iv. Sub-simulators submit updated states of the virtual world to the kernel.
v. The kernel integrates the received states, and sends it to the viewer.

One cycle in the simulation corresponds to a predefined time, this time can be edited by the user to

simulate different cycle's time – it's useful when the RTSCup agent has a long processing time or when the network has significant time delays.

6.3 Simulator Features

The main simulator features are related to the identified problems during the previous comparative analysis of other simulators. The main problems are the lack of focus on AI simulation, while other simulators are not very user friendly (requiring some background knowledge of its implementation to use it).

Client-server architecture: the most common architecture in commercial RTS games is based on the client-side simulation. A more detailed description of this approach can be found at [Boson 2005]. Clients simulate the game actions and they only send the player actions to a central server or to their peers (*peer-to-peer*). An alternative approach is the client-server architecture. In this approach a central server (kernel) simulates the RTS game and only sends, to its clients, the game state that a client must know. This approach is safer because the simulation is carried out into the server so that part of the information can be hidden from clients.

Open communication protocol: the client-server approach, allied to a well-defined and documented communication protocol between clients and server, enables that different clients can be developed using different programming languages. In this way, the researcher can use his previous work without worrying about the programming language that the simulator uses.

Multiplatform server: a multiplatform application enables the server to be executed over several operational systems. This feature guarantees that researchers can use their previous works without worrying about the platform. To ensure this feature in our new simulator version, we are using Java as programming language and a communication strategy based on sockets (UDP).

Parameterized game description: this feature means that the simulator enables a flexible definition of test scenarios. Besides the fact that RTS games have several similar features, they also present some particular ones. The main features that differ from one game to another are: building types, unit types, terrain types, tech-trees types and finish conditions. Our intention is to enable the customization of these and others options, so that users can simulate different game scenarios. This feature is implemented, and is being improved, as a configuration file, which indicates the features (value of parameters) of a particular game. In this way, users can have a library of games, each of them customized to test a particular feature of their approaches.

7. Conclusion and Directions

The main aim of this project is to create a basis for the development of a robust, but user friendly RTS simulator, which could be an alternative to ORTS in the AIIDE game competition. With this goal in mind, some improvements are already in progress, as discussed in next paragraphs.

3D Viewer: we believe that a 3D Viewer for the RTSCup Simulator would be more attractive to general people. But this requirement does not have a high priority, because the main goal can be archived with the current 2D Viewer.

Map editor: the map editor is very useful to enable the creation of different game scenarios, which means that users can have a library of maps to validate their solutions. With this feature, users can verify the adequacy and generalization of the proposed solution (e.g., pathfinding algorithm).

Automatic report: this feature is important to any simulation tool that is being used as a benchmark. The generation of reports enables, for example, the evaluation of solutions in a post-simulation stage, as well as a better basis for comparative evaluations.

Publish the project over the internet: all this documentation will be available in the website (which is currently under development) with all the information that users could need to develop their strategies. This website will also contain some strategy ideas with their source codes, which users can compare with their ones. Our intention is to create a community in this site, with a forum where users can discuss about strategies, exchange experiences, download other client modules (in Java, C, C++, etc) and talk about their doubts.

Creation of a public contest: to support the community interaction, we intend to create public contests, where any people who wants to participate, would send his/her source code and a brief explanation about his/her strategy. The first competition happened in the first semester of 2007, in the Center of Informatics from Federal University of Pernambuco, where groups of students from the course of Intelligent Agents, compete against each other in game 1 and game 2. Their codes are available in the RTSCup's website. This competition was also used to validate the RTSCup.

We are implementing all these improvements considering, at the same time, the usability and documentation from the simulator, from the installation process to the simulation execution. In brief, our objective is to enable that users keep the focus on their research rather than on understanding the simulator operation, or learning a new programming language.

References

BOSON, 2005. *Boson Homepage* [online] Available from: boson.eu.org/ [Accessed 19 August 2007].

BURO, M. 2003. *Real-Time Strategy Games: A new AI Research Challenge*. *In: Proceedings of the 2003 International Joint Conference on Artificial Intelligence*, Acapulco, Mexico.

GLEST, 2007. *Glest Homepage* [online] Available from: www.glest.org/ [Accessed 19 August 2007].

GOMEZ, M. 1999. *Citing references: Simple Intersection Tests for Games* [online] Available from: www.gamasutra.com/features/19991018/Gomez_1.htm [Accessed 19 August 2007].

HANKS, S., POLLACK, M. AND COHEN, P. 1993. *Benchmarks, Testbeds, Controlled Experimentation, and the Design of Agent Architectures. Technical Report 93-06-05*, Department of Computer Science and Engineering, University of Washington, Seattle, WA, USA.

JARTS, 2006. *JaRTS Homepage* [online] Available from: www.cin.ufpe.br/~vvf/jarts/ [Accessed 19 August 2007].

KITANO, H. AND TADOKORO, S. 2001. *RoboCup Rescue: A Grand Challenge for Multiagent and Intelligent Systems*. *AI Magazine*, 22(1):39-52.

LI, S. 2002. *Rock 'em, sock 'em Robocode!*. *IBM developerWorks*, Available from: http://www.ibm.com/developerworks/java/library/j-robocode/.

MCDERMOTT, D. 2000. *The 1998 Planning Systems Competitions*. *AI Magazine*, 4(2):115-129.

ORTS, 2003. *ORTS Homepage* [online] Available from: www.cs.ualberta.ca/~mburo/orts/orts.html [Accessed 19 August 2007].

PONSEN, M., LEE-URBAN, S., MUÑOZ-AVILA, H., AHA, D. AND MOLINEAUX, M. 2005. *Stratagus: An open-source game engine for research in real-time strategy games*. *In: Proceedings of the IJCAI Workshop Reasoning Representation, and Learning in Computer Games*. Edinburgh, UK.

STONE, P. 2003. *Multiagent Competitions and Resarch: Lessons from RoboCup and TAC*. In: Gal A. Kaminka, Pedro U. Lima, and Raul Rojas, editors, RoboCup-2002: Robot Soccer World Cup VI, *Lecture Notes in Artificial Intelligence*, pp. 224–237, Springer Verlag, Berlin.

STRATAGUS TEAM, 2007. *Stratagus: A Real-Time Strategy Engine* [online] Available from: www.stratagus.org/games.shtml [Accessed 19 August 2007].

WELLMAN, M., WURMAN, P., O'MALLEY, K., BANGERA, R., LIN, S., REEVES, D. AND WASH, W. (2001) "A Trading Agent Competition". IEEE Internet Computing, 5(2):43-51.

Utilizando *Rapidly-Exploring Random Trees* (*RRTs*) para o Planejamento de Caminhos em Jogos

Samir Araújo de Souza Luiz Chaimowicz

Departamento de Ciência da Computação
Instituto de Ciências Exatas
Universidade Federal de Minas Gerais

Abstract

Path planning is one of the main components of artificial intelligence algorithms used to control the behavior of NPCs (Non-Player Characters) in games. So far, most of the algorithms rely on detailed information about the game environment and generate paths that are "too perfect", making NPC's behavior sometimes unrealistic. This paper proposes the use of Rapidly-Exploring Random Trees (RRTs) for NPC navigation. Using probabilistic techniques and partial information about the environment, RRTs generate paths that are less artificial, making NPCs behavior more close to human players. Experiments performed using *Quake II* showed the potential of the proposed approach.

Keywords: Path Planning, Rapidly-Exploring Random Trees, *Quake II*.

Resumo

O planejamento de caminhos para NPCs (*Non-Player Characters*) é um dos principais componentes da inteligência artificial em jogos gráficos. A grande maioria dos algoritmos existentes se baseia em um conhecimento detalhado do ambiente e gera caminhos que muitas vezes são "perfeitos demais" tornando o comportamento dos NPCs pouco realista. Esse trabalho propõe a utilização de *Rapidly-Exploring Random Trees* (RRTs) para a navegação de NPCs. Utilizando técnicas probabilísticas e informação parcial do ambiente, as RRTs geram caminhos menos artificiais para os NPCs, equiparando o seu comportamento aos jogadores humanos. Testes realizados utilizando o jogo *Quake II* demonstraram potencial dessa abordagem.

Palavras Chave: Planejamento de Caminhos, *Rapidly-Exploring Random Trees*, *Quake II*.

Contato dos Autores:
{samir,chaimo}@dcc.ufmg.br

1. Introdução

A contínua mudança no perfil dos jogadores demanda, da indústria de jogos computadorizados, investimentos gradativamente maiores em Inteligência Artificial para tornar os seus produtos mais atrativos, divertidos, desafiadores e competitivos [Champandard, 2003]. Atualmente, os jogadores não buscam apenas qualidades nos recursos áudio-visuais. Se um jogo não oferece entretenimento e "desafios inteligentes", adequados ao nível de experiência do seu público alvo, é muito provável que este título não tenha sucesso. Pode-se entender como "desafios inteligentes" desde um quebra-cabeça que leve o jogador à próxima etapa da aventura até um NPC (*Non-Player Character*) [Bourg 2004] que utilize estratégias de movimentação e combates parecidas com as de um jogador humano e seja difícil, mas não impossível de ser vencido.

Em geral, um aspecto importante nos jogos gráficos é a movimentação dos NPCs. A maior parte dos algoritmos para esta finalidade se baseia em estruturas de dados construídas sobre o mapa do ambiente. Estas estruturas, geralmente chamadas de *Roadmaps,* possuem os dados de todos os pontos acessíveis no mapa, os caminhos existentes entre os diversos ambientes que o compõem, além de muitos outros dados valiosos (ex. tipo do ambiente, viscosidade, altura, etc.). Em geral, a maior parte dos jogos utiliza mapas e *roadmaps* ricos em informação, de forma a facilitar os processos de renderização, detecção de colisões, localização e posicionamento dos personagens nos ambientes.

Entretanto, em diversos jogos *multi-player* que implementam NPCs para servirem de oponentes a jogadores humanos ou completarem times de combatentes, estes mapas são utilizados de forma inadequada. A disponibilização de uma grande quantidade de informações aos NPCs, muitas delas não disponíveis ao jogador humano, pode tornar os oponentes imbatíveis e, conseqüentemente, tornará o jogo tedioso, cansativo e sem graça. Desenvolver NPCs desta forma ainda é uma prática comum, principalmente nos jogos de combate em primeira pessoa.

Portanto, o desenvolvimento de aplicações de controle de NPCs que utilizem adequadamente o conjunto de informações disponíveis, sem favorecer quaisquer jogadores (humanos ou virtuais), é um dos grandes desafios na implementação da inteligência artificial em jogos. Nesse contexto, este artigo propõem uma abordagem que permite a geração de caminhos utilizando uma menor quantidade de informações sobre o ambiente. O objetivo é tornar a navegação dos NPCs menos artificial e previsível, mas ainda assim objetiva.

Para isso, são empregadas *Rapidly-Exploring Random Trees* (RRTs), uma técnica comummente utilizada em robótica móvel para o planejamento de caminhos [Lavalle, 1998]. Utilizando técnicas probabilísticas e informação parcial do ambiente, as RRTs geram em tempo de execução possíveis caminhos para os NPCs. Diferentemente dos *Roadmaps* tradicionais que são fixos, a geração de caminhos efetuada pelo RRT possui um certo grau de aleatoriedade, tornando pouco provável que os mesmos caminhos sejam gerados duas ou mais vezes para cada execução do algoritmo em um dado ambiente.

Para a experimentação da abordagem proposta, foi utilizado o jogo Quake II (Figura 1) da ID Software[tm] [ID, 1990]. O Quake II é um jogo comercial, mundialmente conhecido e de código aberto que possui diversos recursos que facilitaram a implementação dos algoritmos.

Figura 1 : Visão do jogador em uma partida *multi-player* de Quake II.

Este trabalho está organizado da seguinte forma: na seção 2 são apresentados os principais trabalhos relacionados. A Seção 3 faz um resumo dos principais conceitos e da constituição das *Rapidly-Exploring Random Trees*, além de apresentar o algoritmo para a sua geração. Na Seção 4 é descrita a implementação do algoritmo no Quake II enquanto na Seção 5 são apresentados os experimentos realizados e seus resultados. Por fim, as conclusões e perspectivas de trabalhos futuros são apresentadas na Seção 6.

2. Trabalhos Relacionados

Apesar de existirem diversas formas de se representar ambientes em jogos (mapa de *tiles*, vetores, polígonos, etc.), as principais técnicas para o planejamento de caminhos em jogos se baseia em grafos: os nós marcam as posições onde o NPC pode passar (em geral chamadas de *waypoints*) e as arestas representam a conexão entre os *waypoints* [Buckland, 2005]. Logo, para cada tipo de mapa, os desenvolvedores de jogos devem criar ferramentas capazes de converter as informações espaciais pertinentes dos mapas nos respectivos grafos. Normalmente, as ferramentas de conversão trabalham de forma automática, mas dependendo do ambiente é exigido dos programadores a inserção manual de nós, para que os caminhos funcionem corretamente e façam sentido [Lent, 2007].

Em geral, o grafo construído a partir dos mapas é chamado de *Roadmap.* Sobre esse *roadmap,* aplica-se algoritmos clássicos de busca de caminhos como A* ou *Dijkstra* de forma a encontrar os melhores caminhos para os NPCs. Como esses algoritmos serão executados diversas vezes durante o jogo, é interessante que que esse grafo seja o mais enxuto possível, de forma a não comprometer a sua eficiência.

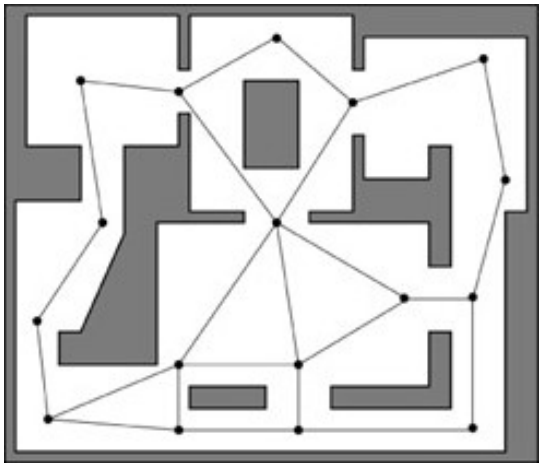


Figura 2 : Roadmap baseado em POV.

Existem diversas técnicas de posicionamento dos nós durante a construção de um *Roadmap*. A Figura 2 apresenta um *Roadmap* que utiliza o conceito de *Points of Visibility* (POV) [Buckland, 2005]. Nessa técnica os nós são posicionados, normalmente de forma manual, em regiões chave do ambiente de forma que mantenham uma conexão com pelo menos algum outro nó. Já uma técnica automática para a construção do *roadmap* são os Grafos de Visibilidade [Lozano-Pérez and Wesley, 1979]. No grafo de visibilidade, os vértices de cada obstáculo são considerados como nós do grafo e são conectados aos vértices de todos os outros obstáculos que possuam caminhos livres entre si, gerando uma malha de conexões por todas as partes do ambiente.

Outras abordagens consistem em dividir o ambiente em células ao invés de se utilizar um grafo. Basicamente, o algoritmo de decomposição em células divide o mapa em áreas poligonais, representando áreas livres ao invés de simples nós de acesso. Em jogos, essa técnica é geralmente chamada de NavMesh [Buckland, 2005].

Apesar da existência de ferramentas para a geração de *waypoints* de forma automatizada, diversos desenvolvedores de jogos optam pela geração manual completa (em todo o mapa) ou parcial (parte manual e parte automática) dos nós. Esta opção é influenciada basicamente pelo formato dos mapas do jogo. Cenários que misturam mapas de altura e ambientes fechados são difíceis de serem mapeados de forma automática. O mapeamento dos *waypoints* se torna ainda mais complicado quando os NPCs possuem seis graus de liberdade para o seu deslocamento (aeronaves, pássaros, submarinos, peixes, etc.). Neste último caso, a geração automática de *waypoints* pode se tornar computacionalmente custosa e ineficiente.

Na verdade, a maioria das técnicas de planejamento de caminhos utilizadas em jogos têm como base os algoritmos utilizados em robótica [Latombe, 1990]. O algoritmo RRT utilizado neste trabalho também foi originalmente proposto para a navegação de robôs móveis [LaValle, 1998]. Como será descrito na próxima seção, o algoritmo constrói um *roadmap* de forma dinâmica, gerando nós aleatoriamente e conectando-os ao conjunto de nós já existente.

Diversos estudos, experimentos e extensões têm sido desenvolvidos com base no algoritmo original. Por exemplo, Zucker e Kuffner [2007] realizaram experimentos utilizando RRTs em ambientes dinâmicos (com obstáculos e plataformas em movimento). Através de adaptações no algoritmo eles conseguiram efetuar o planejamento de caminhos, gerando o *roadmap* diversas vezes em tempo de execução. Dessa forma, a árvore gerada consegue capturar obstáculos móveis e antecipar a necessidade de desvios no percurso em andamento.

Como discutido em [LaValle and Kuffner, 2000], as *Rapidly-Exploring Random Trees* têm um grande potencial e podem ser utilizadas na solução de diversos problemas complexos. O objetivo aqui é explorar essa técnica para o planejamento de caminhos em jogos, uma área na qual essa classe de algoritmos ainda não foi experimentada.

3. Rapidly-Exploring Random Trees

De forma geral, o planejamento de caminhos consiste em encontrar um caminho que leve um NPC de uma posição inicial a uma posição final em um ambiente com obstáculos. Mais formalmente, considera-se que o NPC possui uma determinada configuração $\boldsymbol{c}$ (por exemplo $\boldsymbol{c} = [x,y]$) em um espaço de configurações C (por exemplo R^2). Na verdade, C é dado pela união entre o espaço de configurações livre C_{free} e o espaço ocupado pelos obstáculos C_{obs}. Portanto, o planejamento de caminhos consiste em encontrar um conjunto de configurações em C_{free} que possa levar o NPC de uma configuração inicial $\boldsymbol{c_0}$ até uma configuração final $\boldsymbol{c_f}$. Muitas vezes, não é possível determinar a priori os conjuntos C_{free} e C_{obs}, mas apenas consultar se uma determinada configuração $\boldsymbol{c}$ pertence a um dos conjuntos. Esse tipo de consulta é muito utilizado em algoritmos de planejamento por amostragem (*sampling based planning algorithms*), como por exemplo as RRTs (*Rapidly Exploring Random Trees*).

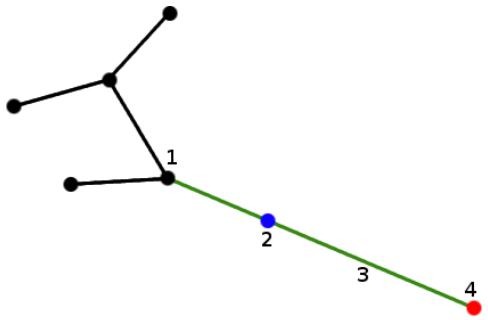


Figura 3 : Etapa de crescimento da RRT.

Dado o espaço de configurações C com seus componentes C_{free} e C_{obs}, e o ponto inicial $\boldsymbol{c_0}$, o algoritmo RRT basicamente tenta cobrir C_{free} com o maior número de caminhos possíveis. A árvore tem a sua raiz em $\boldsymbol{c_0}$, e são realizadas K tentativas de se expandir a árvore pelo ambiente através da incorporação de nós criados randomicamente no espaço livre C_{free}. Cada uma das K tentativas é definida pela execução das etapas abaixo, considerando que o nó inicial, correspondente à raiz, já foi inserido na árvore:

- Utilizando-se uma função randômica, uma posição aleatória dentro do mapa é definida (Figura 3, número 4). Esta posição servirá de orientação para o crescimento da árvore na tentativa vigente.
- É feita uma ordenação nos nós já incorporados à árvore, utilizando-se como métrica a distância em linha reta entre o nó e a posição randômica definida no passo anterior.
- O nó que possuir a menor distância é então escolhido para a tentativa de crescimento da árvore (Figura 3, número 1).
- É traçado um vetor que parte do nó escolhido em direção à posição randômica (Figura 3, número 3). Contudo, somente uma parte do vetor será utilizada, pois existe uma constante que limita o tamanho máximo que um galho pode crescer. Esta constante define o tamanho do segmento do vetor que será avaliado, conforme visto na Figura 3, número 2. O segmento parte do nó candidado na cor preta (1) e vai até o ponto azul (2).

- Se o segmento estiver contido inteiramente em $C_{free,}$ o nó definido pelo ponto azul é então incorporado à árvore, caso contrário a árvore não muda.

O número *K* é definido a priori e vai acabar determinado o tempo de execução do algoritmo e a sua cobertura no ambiente (quanto maior o *K*, maiores serão as chances da árvore crescer e cobrir o ambiente, porém mais demorada será a execução). Quanto mais homogênea a distribuição dos nós da árvore pelo espaço livre do ambiente, mais eficiente terá sido a execução do algoritmo. Ao final das *K* tentativas de crescimento, o algoritmo é encerrado e tem-se como resultado uma árvore, cujas arestas formam caminhos em potencial e os nós representam pontos distintos do ambiente.

Algoritmo 1: Geração de *Roadmap* utilizando RRT

```
1.  GENERATE_RRT(c0 , K, T)
2.     T.inicio(c0 );
3.     Para k=1 até K faça
4.        // busca uma posição randômica no mapa
5.        crand ← EstadoRandomico( )
6.        // encontra o vizinho mais próximo de Crand
7.        cproximo ←  VizinhoMaisProximo(T, crand )
8.        // tenta crescer a árvore, evitando colisões
9.        u ← selecionaEntrada(crand, cproximo)
10.       cnovo ←  novoEstado(cproximo, u, T)
11.       // conecta o novo nó à árvore
12.       T.adiciona(cnovo)
13. Retorna T
```

A randomização é uma técnica muito comum em algoritmos probabilísticos e tem grande valor em vários contextos. O algoritmo RRT citado nesta seção utiliza-se da randomização para rapidamente explorar o ambiente. É correto afirmar que a randomização é um fator essencial para sua eficiência [LaValle, 2006].

São listadas abaixo algumas das propriedades e características interessantes do algoritmo RRT:

- A expansão da RRT é guiada para espaços ainda não explorados.
- Uma RRT é probabilisticamente completa.
- O algoritmo RRT é relativamente simples, o que facilita a análise de desempenho.
- A RRT sempre permanece conectada, independente da quantidade de vértices, mesmo que esses sejam mínimos.
- Uma RRT pode ser considerada um algoritmo de planejamento de caminhos e pode ser adaptada para uma grande quantidade de sistemas.

A Figura 4 mostra duas etapas do crescimento de uma *Rapidly Exploring Random Tree* em um ambiente inicialmente vazio.

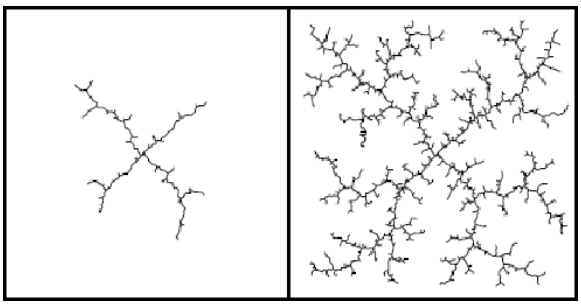


Figura 4: Crescimento de uma RRT.

É possível observar na Figura 4 que a RRT rapidamente se expande em várias direções para explorar os quatro cantos do ambiente. Apesar do algoritmo ser relativamente simples, os resultados de sua execução são bastante satisfatórios em diversas situações, principalmente em ambientes com poucos obstáculos e passagens não muito estreitas.

Não é difícil provar que os vértices serão distribuídos uniformemente no espaço. No começo da expansão da árvore, claramente não se trata de uma distribuição uniforme, entretanto à medida que o tempo avança os pontos randômicos são gerados e os vértices se tornam uniformes. Esse resultado é independente da localização da raiz. Além disso, a RRT é mínima no sentido que sempre mantém os vértices conectados. A detecção de obstáculos, que sempre foi um processo lento para algoritmos de planejamento, não é um problema para as RRTs, pois a detecção pode ser feita de forma incremental.

Todas essas características das RRTs motivaram o desenvolvimento deste trabalho pois indicam, de forma intuitiva, que esta ferramenta pode perfeitamente ser utilizada em jogos computadorizados.

4. Implementando RRTs no Quake II

Para testar um algoritmo de planejamento de caminhos que gera as trajetórias para os NPCs dinamicamente e em tempo real, é necessário um ambiente adequado de experimentação. De preferência um jogo comercial que facilite a implementação de novas funcionalidades. Como um laboratório de testes, o Quake II foi utilizado no processo de experimentação do algoritmo de planejamento de caminhos RRT.

Como o Quake II não possui recursos nativos para a utilização de NPCs no modo *multi-player*, diversos programadores independentes escreveram diferentes modificações (MODs) [Holmes, 2002] para o jogo. Estes MODs disponibilizam aos jogadores a opção de utilizar NPCs como seus oponentes. O MOD utilizado no desenvolvimento deste trabalho se chama Jabot [Jabot, 2005]. Diversos MODs foram avaliados, mas o Jabot possui recursos de navegação que facilitam a injeção de funcionalidades extras, além do seu código ser totalmente aberto e livremente distribuído pelo autor.

O Quake II utiliza o BSP [Brackeen et al. 2003] como estrutura de dados para a representação de seus mapas. A natureza do BSP permite a extração de informações detalhadas sobre os elementos que constituem o mapa e aqueles que estão inseridos nele (ex. personagens, itens, obstáculos em movimento, etc.). O MOD de controle dos NPCs tem acesso completo às estruturas de dados e funcionalidades do jogo. As entidades que representam os personagens, os itens do jogo e funções de detecção de colisão são algumas das funcionalidades requeridas para se programar NPCs no Quake II. Entende-se como entidades, estruturas de dados que armazenam as características do elemento, tais como posição no mapa, dimensões, texturas, etc. Criar programas de controle de NPCs mais divertidos e menos artificiais, que utilizam ponderadamente estes recursos, é um grande desafio, mas deveria ser um dos principais objetivos dos desenvolvedores de sistemas de Inteligência Artificial em jogos.

O Jabot utiliza para a navegação dos NPCs um arquivo que contém uma árvore. Os nós desta árvore correspondem a pontos no mapa, que representam áreas acessíveis aos jogadores. As arestas atuam como caminhos livres que conectam as áreas entre si. Esta estrutura sempre é utilizada para orientar o deslocamento dos NPCs entre dois pontos no mapa.

O preenchimento do arquivo de navegação efetuado pelo Jabot é feito da seguinte forma:

- As entidades do mapa são lidas e identificadas.
- Para cada entidade do mapa, um nó é lançado no arquivo indicando o seu tipo.
- Quando uma partida é iniciada, o caminhamento executado pelos jogadores humanos é mapeado através da inclusão de novos nós e arestas à árvore, aumentando a malha de alcance dos NPCs.
- Inicialmente, como a árvore é pequena, os NPCs somente se deslocam no modo *Wander* (vagueando aleatoriamente pelo cenário), mas seus caminhamentos também são mapeados na árvore, assim como os dos jogadores humanos.
- Após a execução de diversas partidas, o número de nós atinge um limite e então a árvore é considerada completa.

Utilizando parte do código-fonte do Quake II, construiu-se uma ferramenta para a geração do arquivo de navegação no formato original do Jabot. Esta ferramenta possibilitou a utilização do RRT na geração da árvore de navegação dos NPCs. Além desta ferramenta, o MOD foi alterado em diversos pontos para a realização dos experimentos. Entre outras funcionalidades, um console externo foi acoplado ao jogo para a visualização das posições dos NPCs no mapa vigente.

O mapeamento dos caminhos percorridos pelos NPCs durante as partidas foi justamente capturado por este console. A máquina de estados finitos dos NPCs foi reduzida a apenas um comportamento: caminhar pelo ambiente. Esta adaptação foi essencial para a realização dos testes, ao condicionar os NPCs a percorrerem incessantemente os caminhos gerados pelo planejador. Uma funcionalidade incorporada ao MOD foi um comando customizado interpretado pelo console do jogo. Este comando executa o planejador de caminhos e disponibiliza aos NPCs o *Roadmap* diretamente em memória, sem a necessidade de geração de arquivos. O Jabot carrega o seu *Roadmap* em memória utilizando duas listas de elementos, contendo nós e arestas, respectivamente. Então basta preenchê-las com a árvore gerada pelo RRT e os NPCs terão novos caminhos disponíveis. A utilização deste recurso evita a necessidade da geração do arquivo de navegação com uma ferramenta externa, toda vez que o usuário/desenvolvedor desejar alterar as rotas de navegação dos NPCs.

Ao se modificar a máquina de estados finitos dos NPCs, suprimiu-se algumas das funcionalidades previamente programadas. Por exemplo, os NPCs não entram em modo de combate e não atiram em seus oponentes. Seu modo de atuação agiliza a execução dos testes, pois os NPCs buscarão o tempo todo caminhos já planejados em memória para serem percorridos. Dessa forma, para cada versão da árvore de caminhamento gerada pelo algoritmo RRT, trajetos curtos (áreas vizinhas) e longos (áreas distantes) foram realizados com pontos de início e fim distintos e em diversas partes dos cenários.

Apesar do algoritmo RRT poder ser utilizado em ambientes tridimensionais complexos, optou-se, por uma questão de simplificação do contexto, por validá-lo apenas em ambientes planos. Dessa forma, a árvore gerada pelo RRT se encontra completamente em um mesmo nível de altura. Foram criados dois mapas de testes com a ferramenta de edição de mapas do próprio jogo, sem desníveis no piso, cujos diagramas podem ser visualizados nas Figura 5 e 6. O mapa da Figura 5 é constituído de duas salas que se comunicam através de uma passagem estreita. Já o mapa da Figura 6 é um ambiente bastante complexo, contendo um labirinto de passagens estreitas e alguns caminhos sem saída.

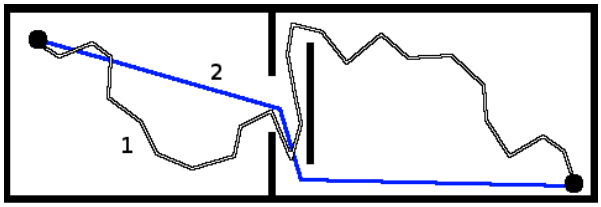


Figura 5 : Mapa 1 (simples). O caminho 1 foi gerado pela RRT enquanto o caminho 2 foi gerado pelo planejador tradicional.

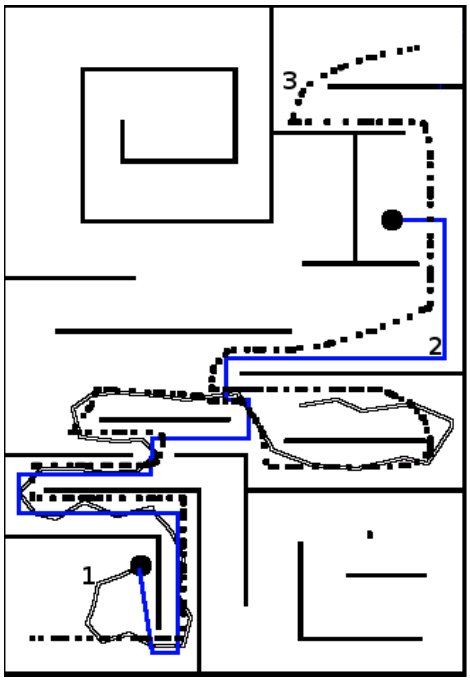


Figura 6 : Mapa 2 (complexo). O caminho 1 foi gerado pela RRT enquanto o caminho 2 foi gerado pelo planejador tradicional. O caminho 3 é o resultado de um caminhamento efetuado por um jogador humano.

É importante ressaltar que, em geral, os caminhos gerados pelo algoritmo RRT não devem ser utilizados diretamente pelo planejador de caminhos dos NPCs. Por serem caminhos gerados dinamicamente, a probabilidade do resultado final ser um conjunto de percursos bastante sinuosos é grande. Caso utilizado diretamente, os NPCs poderiam agir como se estivessem perdidos ou tontos, ao fazerem diversos "zigue-zagues". Dessa forma, recomenda-se o uso de algum algoritmo de pós-processamento para realizar a suavização dos caminhos produzidos pelo RRT. Entretanto, caso o desenvolvedor não deseje usar um algoritmo de pós-processamento, também é possível utilizar alguma heurística para orientar o crescimento da RRT e forçar a geração de caminhos menos sinuosos.

5. Resultados

Como os experimentos realizados abordam principalmente questões comportamentais dos NPCs, especificamente a forma com que eles se movimentam pelo ambiente, os principais aspectos observados nos experimentos foram o desenho do caminho gerado por cada planejador e se o alvo foi alcançado ou não.

Os experimentos foram realizados com K = 600 para o mapa 1 e K = 2500 para o mapa 2. O tempo médio de execução em ambos os cenários foi 0.5 segundos para o mapa 1 e 4 segundos para o mapa 2, utilizando um *Pentium M* 1.6 Ghz. Conforme descrito na Seção 3, quanto maior o valor de K maior será o tempo de execução demandado, independentemente do tamanho do ambiente de execução.

Sem a utilização de mecanismos de suavização de caminhos, são apresentados nas Figuras 5 e 6 trajetos gerados pelos planejadores de caminhos tradicional e RRT em ambos os mapas de testes. Além disso, na Figura 6, também é mostrado um caminho gerado por um jogador humano controlando o personagem. Os pontos pretos representam o início e o fim do percurso. Em ambos os mapas, o trajeto em azul - gerado pelo planejador de caminhos que utiliza as informações do mapa (privilegiando os NPCs) - sempre é ótimo e completo. Ou seja, é o caminho mais curto entre os pontos de início e fim, além de sempre ser uma solução que atinge o objetivo, mesmo sem o NPC ter explorado ou percorrido o ambiente alvo pelo menos uma vez na partida.

Os caminhos gerados pelo planejador que utiliza o algoritmo RRT têm um comportamento bastante diferente do gerado pelo planejador tradicional. No mapa da Figura 5, apesar do objetivo ter sido alcançado, o caminho gerado é sinuoso e distante do ótimo, representando um possível atraso na chegada do NPC ao ponto de destino. Como a árvore pode ser melhorada durante a partida (com a incorporação de novos nós) os próximos caminhos planejados para estes mesmos pontos de início e fim tenderão a se aproximar do ótimo, mas nunca alcançá-lo. Esta incorporação de nós à árvore pode representar o aprendizado dos NPCs em relação ao ambiente. Já no mapa da Figura 6, o caminho resultante apresentado demonstra um cenário cuja árvore não foi capaz de cobrir toda a extensão do mapa, ocasionando um planejamento de caminhos incompleto. Já este comportamento pode ser visto como um fator de equilíbrio, dado que diversos ambientes do cenário serão, no início da partida, desconhecidos pelos NPCs assim como pelos jogadores humanos.

Também é interessante observar que o caminho executado por um jogador humano é de certa forma intermediário entre os dois caminhos planejados. Em regiões estreitas, o jogador tem um caminho semelhante ao ótimo, exatamente por se tratar de regiões onde não há muitas opções a não ser seguir em linha reta. O RRT, entretanto, gera trajetórias sinuosas, aspecto que pode ser corrigido utilizando um pós-processamento como mencionado na seção anterior. Já em regiões mais abertas, onde é necessário explorar o ambiente, o jogador humano tem um comportamento similar ao RRT, o que de certa forma demonstra um dos objetivos de se usar RRTs neste trabalho que é gerar caminhos menos artificiais para o NPC.

A árvore completa gerada pelo algoritmo RRT para a realização dos caminhamentos no mapa da Figura 6 pode ser visualizada na Figura 7. Como mencionado, como o número de tentativas (K) não foi suficientemente grande, o algoritmo não foi capaz de explorar o ambiente adequadamente e com isso não encontrou um caminho entre o ponto inicial e o objetivo.

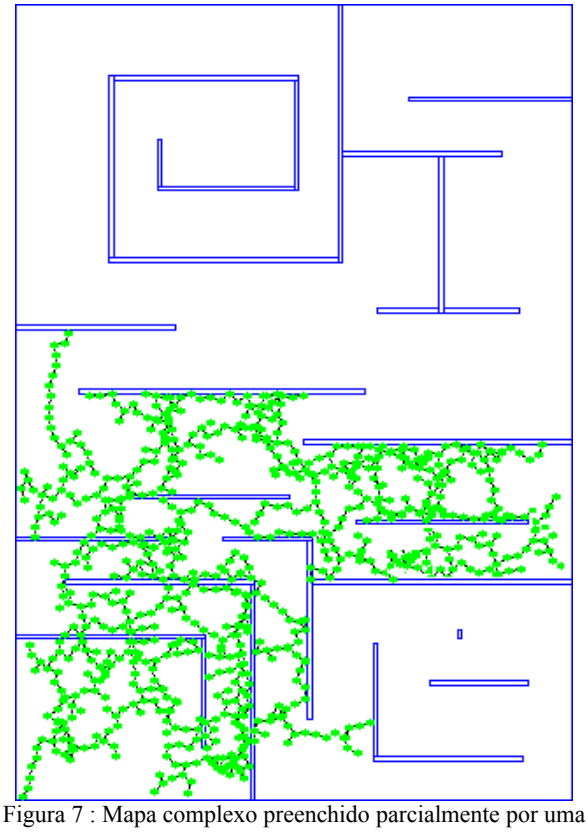


Figura 7 : Mapa complexo preenchido parcialmente por uma árvore gerada pelo RRT, utilizando $K = 2500$.

Os resultados obtidos neste experimento levam à conclusão que o RRT é um algoritmo eficiente para o planejamento de caminhos, desde que utilizado corretamente. A execução do RRT no mapa 1 mostrou que ambientes pequenos são fáceis de serem cobertos pela árvore gerada pelo RRT. Isto reforça ainda mais a idéia de se gerar pequenas árvores, para cobrir uma ou duas salas de cada vez, ao invés de tentar mapear o mapa por completo. Já os experimentos realizados no mapa 2 mostraram que é necessário efetuar adaptações no algoritmo para conduzir o crescimento da árvore de forma mais eficiente em ambientes muito extensos e com muitos caminhos falsos. Talvez uma abordagem de geração de pequenas árvores ao longo de uma caminhada, que serão interconectadas para a constituição do *Roadmap*, seja mais interessante para esse tipo de ambiente.

A utilização do RRT introduz incerteza no mapeamento dos possíveis caminhos existentes no ambiente. Logo, o NPC não tem mais acesso a informações precisas sobre o mapa e passa a ter apenas uma noção do ambiente, assim como um jogador humano. Utilizar caminhos gerados pelo RRT pode tornar os NPCs mais vulneráveis, caso os caminhos gerados pelo algoritmo sejam incompletos ou distantes do ótimo. Nestes casos, para evitar que jogadores humanos sempre tenham vantagens sobre os NPCs, pode-se utilizar mecanismos de planejamento de caminhos locais. Variações do próprio RRT podem ser utilizadas para executar esta atividade.

6. Conclusões

Dados os resultados dos experimentos realizados neste trabalho, pode-se dizer que o algoritmo RRT se comportou como uma ferramenta interessante e viável para o planejamento de caminhos para NPCs. Em geral, a sua implementação é simples e não requer funcionalidades adicionais àquelas de detecção de colisões já existentes no jogo. Sua utilização em jogos pode contribuir para aumentar o fator de diversão, ao remodelar os caminhos dos NPCs de forma imprevisível. Por se tratar de um algoritmo randômico, é impossível prever qual será a extensão do cenário coberta pela árvore gerada pelo RRT. Basicamente, os caminhos gerados podem ter diversas formas e conectar ou não dois pontos de um possível trajeto. Portanto, a RRT pode ser vista como um recurso para se gerar *Roadmaps* aleatórios a cada execução do jogo, tornando o comportamento dos NPCs menos previsível e mais susceptível a falhas. Os *Roadmaps* gerados pelo RRT são incertos se comparados aos *Waypoints* e grafos topológicos geralmente utilizados em jogos. E é justamente esta incerteza que representa o ganho em diversão para o jogador.

Além disso modificações podem ser feitas no algoritmo, para adaptá-lo ao estilo de jogo e ao perfil de seus jogadores. Reduzir o alcance da árvore gerada e efetuar o seu reprocessamento diversas vezes, pode gerar um NPC que cria o seu *Roadmap* e faz o planejamento de caminhos curtos de forma mais eficiente. Esta é uma dentre as inúmeras possibilidades de adaptações no RRT que podem ser realizadas para se alcançar os mais diversos comportamentos dos NPCs.

Os mapas do Quake II são normalmente formados por ambientes fechados, contendo salas conectadas entre si. Os experimentos realizados demonstraram que o algoritmo RRT se comporta bem neste tipo de ambiente, com desempenho satisfatório ao ser executado em tempo real. Para experimentos futuros, deseja-se explorar outros tipos de ambientes, por exemplo locais abertos, com vários desníveis no solo e de preferência mesclando diversos tipos de estruturas de dados como mapas de altura, BSP, entre outros.

Por fim, é importante mencionar que a utilização desta técnica em jogos onde o formato do mapa não possua tantas informações quanto o BSP pode ser bastante promissora. A execução do RRT não necessita de informações completas do ambiente e requer apenas uma descrição dos seus limites e dos obstáculos nele contidos. Isso faz com que o algoritmo se torne especialmente adequado em jogos com estruturas de dados mais simples. Mas mesmo em ambientes ricos em informação, nota-se com o desenvolvimento deste trabalho que também é possível utilizar técnicas que não se beneficiam dessas informações privilegiadas com o objetivo de produzir NPCs mais versáteis e menos artificiais.

Referências

BOURG, M., Seeman, G., 2004. AI for Game Developers. O'Reilly Press.

BRACKEEN, D., BARKER, B., VANHELSUWÉ, L. 2003. Developing Games in Java. New Riders Publishing.

BUCKLAND, M., 2005. Programming Game AI by Example. Worldware Publishing.

CHAMPANDARD, A. J., 2003. AI Game Development: Synthetic Creatures with Learning and Reactive Behaviors. New Riders Publishing.

HOLMES. S. 2002. Focus on Mod Programming in Quake III Arena. Thomson Course Technology.

ID. 1990. ID Software (Game Developer). Disponível em: http://www.idsoftware.com/ [Acesso em 18 de Agosto de 2007].

JABOT. 2005. Quake II Bot Modification. Disponível em: http://jal..quakedev.com/jabot.html [Acesso em 18 de Agosto de 2007].

LATOMBE, J. C., 1990. Robot Motion Planning. Kluwer Academic Publishers, Boston, MA.

LAVALLE, S. M., 1998. Rapidly-exploring Random Trees: A new tool for path planning. TR 98-11, Computer Science Dept. Iowa State University.

LAVALLE, S. M., 2006. Planning Algorithms. Cambridge University Press.

LAVALLE, S. M., KUFFNER, J. J 2000. Rapidly-Exploring random trees: Progress and prospects. In *Proceedings of the Workshop on the Algorithmic Foundations of Robotics*.

LENT, M. V. 2007. Game Smarts. IEEE Entertainment Computing [online]. Disponível em: http://www.computer.org/ [Acesso em 18 de Agosto de 2007].

Lozano-Pérez, T., Wesley, M., 1979. An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles. *Communications of ACM* 22(10):560-570.

ZUCKER, M., Kuffner, J.J. 2007. Multipartite RRTs for rapid replanning in dynamic environments. *In Proceedings of Robotics and Automation (ICRA'07),* 2007.

Reflex – Um motor de jogos em Flash

Rodrigo M. A. Silva & Laércio Ferracioli
Laboratório de Tecnologias Interativas Aplicadas a Modelagem Cognitiva
Universidade Federal do Espírito Santo – UFES
Vitória-ES, Brasil

Abstract

This paper shows the design, implementation and use of the Reflex, a Flash Game Engine. The main objective of it is to facilitate the development of 2D Games, using an optimized side-scrolling techniques and physics simulations. To show it features, a simple game was created and will be argued in the end of the article.

Resumo

Este artigo apresenta o projeto, desenvolvimento e uso da Reflex, um Motor de Jogos em Flash. O principal objetivo desse motor é facilitar o desenvolvimento de jogos 2D, utilizando técnicas otimizadas de side-scrolling e simulação de física. Para demonstrar as funcionalidades da mesma, foi criado um jogo exemplo que será discutido no fim do artigo.

Palavras-chave: Desenvolvimento de jogos, Engrenagens de jogos, Flash, jogos 2D.

Contato:
rodrigomas85@yahoo.com.br
laercio.ufes@gmail.com

1. Introdução

A área de desenvolvimento de jogos eletrônicos tem apresentado um grande interesse e crescimento tanto no âmbito comercial, quanto no âmbito acadêmico [COS05] devido ao fato da utilidade dos jogos nos mais variados contextos, tais como, simuladores, sistemas de visualização e outros.

Nesse contexto, durante o desenvolvimento de jogos educacionais, como o Alerta Verde [SIL06], a equipe do ModeLab percebeu que muitas das funcionalidades necessárias para produzir um jogo poderiam ser agrupadas e generalizadas para o desenvolvimento de jogos e aplicações e, desta forma, otimizar o processo de produção.

Dessa análise, o ModeLab desenvolveu a Reflex, um motor de jogos 2D[1] que permite ao desenvolvedor, concentrar-e na idéia do jogo, na elaboração e construção de cenários e de personagens. O projeto foi realizado de forma a oferecer serviços ao usuário, fazendo referência a arquitetura de serviços de rede, simplificando o processo de aprendizagem e utilização. Dentre esses serviços, pode-se destacar: Detecção de Colisão, Resolução de Colisão (e.g. cálculos de recuo), Side-Scrolling entre outros.

2. Desenvolvimento de Jogos

O desenvolvimento moderno de jogos, cada vez mais, necessita de estratégias de desenvolvimento de software, isso devido à sua própria natureza multi e interdisciplinar e ao fato da grande espera, ou exigência, de um alto grau de realismo, tanto gráfico como de simulação [FEI06], mesmo em jogos mais simples, como os 2D. Assim sendo, é importante a utilização de metodologias de Engenharia de Software [PRE05], para tornar viável a produção e manutenção de um jogo.

Além desse fato, observa-se que devido ao forte crescimento da internet [MAG05], muitas aplicações tradicionais vêm sendo substituídas por aplicações web [FRA99], fato que se deve à redução dos custos de acesso aos computadores, aos incentivos à inclusão digital e a mudanças culturais ocorridas. Essa mudança de cenário de desenvolvimento incentiva a criação de *WEB Games* tanto 2D quanto 3D.

Outro aspecto é o grande crescimento de jogos para sistemas embarcados e as perspectivas de aumento desse nicho. Por isso, o uso de ferramentas que dão suporte ao desenvolvimento nessas plataformas é desejável.

Baseado nesse contexto, a Reflex foi elaborada para acompanhar essas tendências e, para isso, foi necessário pesquisar uma ferramenta que incorporasse tais fatores. Depois de uma análise das ferramentas [SAN06] disponíveis no mercado, o Adobe Flash [ADO06] foi escolhido. O principal fator de escolha foi a popularidade do mesmo para desenvolvimento web, a interface de fácil aprendizagem e a portabilidade fornecida por essa ferramenta.

2.1. Ambiente Adobe Flash

O Adobe Flash, ou Flash, é um ambiente de desenvolvimento integrado, IDE - ***I**ntegrated **D**evelopment **E**nvironment*, ou seja, ele possui um ambiente com várias ferramentas para desenvolvimento, tais como, editor de cenas, editor de código, depurador de erros, além de vários outros recursos.

O Flash foi projetado originalmente para ser um ambiente de criação de filmes vetoriais que fossem pequenos e dinâmicos e que possibilitassem uma

[1] *Os termos "jogo" e "game" são usados indistintamente ao longo do texto.*

interação com os usuários[2] finais. Com o tempo, o Flash foi incorporando novos recursos de programação, interface com usuário e animação mais sofisticados, que trouxeram a ferramenta a um patamar de produção de grandes softwares.

Devido ao seu foco inicial, o Flash trabalha com uma interface de criação de filmes, dessa forma, ele divide um projeto em Cenas e Quadros: tal abordagem é especialmente interessante na produção de jogos, pois podemos entender um jogo como sendo um filme interativo. Contudo, deve-se ter uma conduta de desenvolvimento mais rígida e organizada para não tornar a manutenção de um projeto muito difícil.

A partir da versão 8, o Flash permite a utilização de duas linguagens de programação, o ActionScript e o Javascript/ECMAScript [ECM99]. O Javascript é uma linguagem que, normalmente, é interpretada pelo computador. Ela suporta orientação a objetos [BOO94] e é muito utilizada na codificação de programas client-side[3] de páginas web.

O ActionScript é uma linguagem fortemente baseada no Javascript, com suporte a orientação a objetos, programação guiada por eventos e suporte a DOM - ***D**ocument **O**bject **M**odel* [WOR05]. DOM é uma especificação do W3C - ***W**orld **W**ide **W**eb **C**onsortium*, atualmente o maior órgão de padronização da internet, que fornece meios de acesso a dados, independente de linguagem e plataforma. Ele permite ao Flash comunicar-se com linguagens como *Extensible Style Language* (XSL) [WOR06], eXtensive Markup Language (XML) e outras, podendo assim criar aplicações mais ricas e dinâmicas. É possível fazer comunicação do Flash com banco de dados.

Um dos conceitos mais importantes do Flash é o de *MovieClip*. O MovieClip pode ser entendido como um sub-filme do Flash, logo, pode-se pensar que um filme Flash é composto de vários sub-filmes, ou MovieClips, e assim sucessivamente. Essa estrutura é muito útil para produção de elementos hierárquicos, como personagens.

2.1.1. Arquitetura de Programação do Flash

Conforme mencionado, os aplicativos, ou também filmes, gerados pelo Flash são multi-plataforma. A forma utilizada para conseguir tal característica é a utilização de uma Máquina Virtual e, dessa forma, a compilação realizada pelo Flash gera um código intermediário, denominado *bytecode*, que, analogamente às arquiteturas de computação tradicionais, representa o Assembly, ou código de máquina, dessa máquina virtual. Essa metodologia é muito utilizada nas aplicações Java [GOS06] e apresenta prós e contras conforme Tabela 01.

Tabela 01: *Características do Flash*

Vantagens	Desvantagens
Portabilidade	Redução do Desempenho
Redução de Memory Leaks[4]	Necessidade de possuir a Máquina Virtual.
Tratamento de Erros	Funcionamento dependente da Máquina Virtual[5].

Para utilizar um aplicativo desenvolvido no Flash, é necessário ter instalado o Adobe Flash Player, que nada mais é que a Máquina Virtual do Flash. O Adobe Flash Player é gratuito, pequeno e está presente na maioria dos computadores pessoais atuais.

3. Reflex

A Reflex é um motor de jogos que tem como objetivos proporcionar serviços que simplifiquem o processo de criação, modificação e teste de jogos, de forma a não só aperfeiçoar esses processos, como também aumentar a eficiência do jogo, e ainda, prover uma melhor separação da modelagem e da visualização, aumentando a possibilidade de trabalhos paralelos, além de prover uma melhor utilização de recursos.

Conforme apresentado na seção 2.1, o Flash possui alguns problemas de desempenho. Contudo, alguns deles já vêm sendo resolvidos pelas novas versões dos players.

Quando é necessário utilizar side-scrolling, em um cenário que possui muitos MovieClips, mover todos torna o processo extremamente ineficiente, além de consumir muita memória. A Figura 01 mostra o decaimento de desempenho do Flash com o aumento do n° de movieclips e o desempenho usando a estratégia da Reflex. As barras verticais representam erro de medida, obtido via desvio padrão. Nessa figura, a melhoria proporcionada pela Reflex se deve à forma de deslocamento dinâmico de MovieClips na memória, apresentada em 4.2.

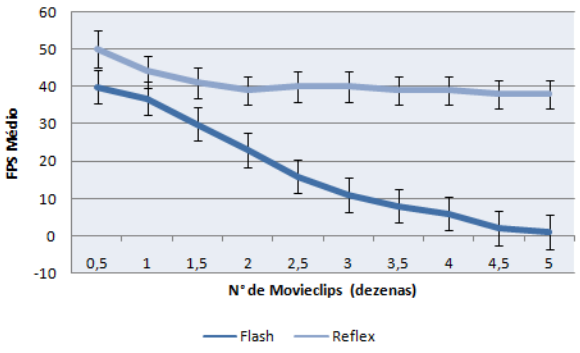


Figura 01: *Desempenho do Side-Scrolling no Flash*

[2] *O termo usuário refere-se aos desenvolvedores de jogos que utilizam o motor, enquanto jogador refere-se aos usuários finais.*

[3] *Programas que são executados pelo computador do usuário final.*

[4] *Desperdícios de memória normalmente ocasionados por erros de programação.*

[5] *No caso de Sistemas Unix-Like, há várias implementações do Flash.*

Para o experimento da Figura 01, foi construído um pequeno cenário no estilo do jogo Super Mário Bros, que pode ser visto na Figura 04, com Side-Scrolling e feito medições em um mesmo computador, no caso um Intel® Pentium 4 1.8 GHz com 512 MBytes de memória RAM. Observando o experimento, constata-se que isso é um problema crítico no Flash, pois a maioria dos jogos necessita de side-scrolling. O experimento mostra ainda que a estratégia utilizada pela Reflex consegue manter um bom nível de FPS[6].

Além dessa caracterísitica, o motor oferece outros serviços, tais como, o de Download de MovieClips em tempo de execução, tratamento de erros, detecção de colisão.

Concluindo, a Reflex não se preocupa em como funciona a animação dos personagens, a inteligência e movimentação deles: ela simplesmente realiza os serviços sobre eles, idéia que será explicitada na seção 5.

4. Projeto e Desenvolvimento da Reflex

O Projeto do motor Reflex foi idealizado usando a orientação a objetos e o conceito de camadas [DAY83]. Dessa forma pode-se dividir as tarefas de uma forma mais independente, bastando estabelecer uma interface padrão entre as camadas. Outra vantagem é que se houver necessidade de alterar uma subcamada, as demais camadas não necessitarão de passar por alteração. A Figura 02 mostra o diagrama da arquitetura de camadas da Reflex.

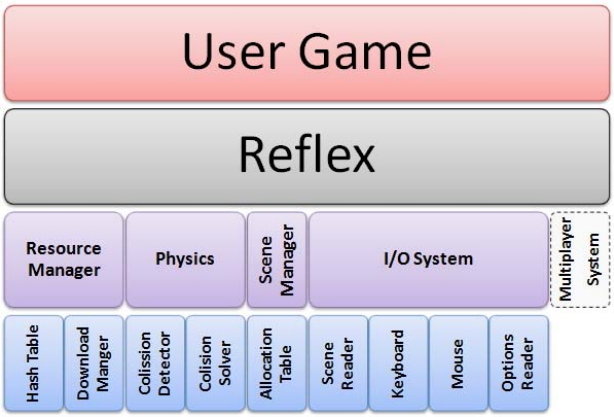


Figura 02: *Camadas do Framework da Reflex*

O motor utiliza recursos do Adobe Flash, contudo, decidiu-se criar uma forma de interagir com o Flash sem utilizar diretamente os recursos desse ambiente: esse fato aumenta a possibilidade de adaptação do motor para outras plataformas. Para isso foi criado um wrapper, ou seja, um intermediador entre o Flash e a Reflex.

Todos os movieclips do Flash ao serem adicionados à Reflex são encapsulados com esse wrapper e são manipulados através dos métodos e propriedades disponibilizadas. Esse foi chamado de ReObject, sendo a principal classe do motor.

4.1. Funcionamento Geral

A Reflex trabalha com a seguinte proposta: o usuário especifica uma cena, usando os movieclips do Flash, na seqüência carrega essa especificação no motor usando os comandos de carregamento e, nesse ponto, a Reflex passa a cuidar de todo o mapeamento real da cena na tela, bem como a Física especificada pelo usuário.

Essa especificação é feita através de um arquivo XML, sendo escolhida pelo usuário a forma de colisão de um movieclip, a posição do mesmo, a forma de carga, entre outros parâmetros.

4.2. Resource Manager

O Resource Manager é o subsistema responsável por armazenar e disponibilizar os recursos do motor. O serviço prestado por ele deve utilizar estruturas de dados apropriadas para armazenamento e acesso rápido aos recursos.

Todos os Movieclips são armazenados em uma Tabela Hash [COR02], esse mapeamento é feito utilizando o conceito de *Virtual Viewports*. Essas estruturas são mostradas na Figura 03.

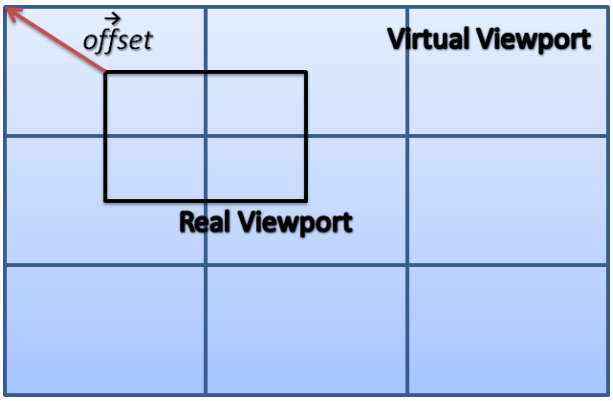


Figura 03: *Mapeamento de MovieClips em Viewports*

O cenário do jogo, normalmente, é muito maior que a área de visualização disponível, sendo assim, o espaço virtual de visualização, que consiste no cenário completo, é dividido em várias viewports, ou regiões. O tamanho dessas é, por padrão, o mesmo da área de visualização real, contudo, é possível especificar outros tamanhos.

Ao carregar o arquivo XML da especificação da cena, a Reflex faz as divisões, e então mapeia todos os movieclips para as viewports correspondentes, esse mapeamento é feito com o uso de uma função de mapeamento, ou função de hash. Através desse processo a Tabela Hash é preenchida, de forma que dada uma virtual viewport obtenha-se todos os objetos

[6] *FPS: Quadros por segundo*

contidos nela. É importante observar que, dessa forma, cada linha da tabela representa uma viewport e ela possui um vetor de objetos.

Assim sendo, quando há um side-scroll, o motor define quais viewports devem ser mostradas e, então, com o uso da Tabela Hash, mostra todos os movieclips que estiverem nessas viewports, evitando assim, problemas relativos a não exibição de alguns movieclips.

Quando uma viewport sai da visualização, os movieclips pertencentes a ela são removidos, de forma a liberar memória. Quando uma viewport aparece pela primeira vez, os movieclips referentes a ela são criados e mantidos até a mesma sair da tela de visualização.

A especificação XML permite a criação de tile based games.

4.2.1. Download Manager

Quando o usuário especifica o tipo de carregamento do movieclip, ele está informando a forma com que o motor vai obter o mesmo. Se o usuário informar que é um carregamento *estático*, o motor buscará o movieclip na biblioteca de movieclips do Flash, ou seja, ela usará um movieclip compilado juntamente com o motor. Obviamente essa é a forma mais rápida de se obter um movieclip.

O usuário pode especificar que um movieclip não está no arquivo do Flash, e o motor deve baixá-lo de um link. São permitidos 3 formas de download.

Uma forma é a *Forçada*, quando o motor deve baixar o movieclip antes de exibir a cena. Outra forma é por *Demanda*, que informa para o motor que somente deve baixar o movieclip, se o jogador chegar à viewport que o contém, ou se for solicitado explicitamente pelo programador. Esse método pode ser entendido como o usado no Google Maps e WikiMapia, nesse modo a cena é exibida sem que o movieclip esteja disponível.

Contudo, a forma mais interessante é a *Contínua*. Ela também permite que a cena seja exibida sem que o movieclip já esteja disponível, visto que, ele vai sendo baixado durante a execução da cena. Esse recurso é muito útil quando se tem grandes cenários, de forma que, partes mais distantes do início da fase possam ser carregadas enquanto o jogo vai ocorrendo, reduzindo, assim, o tempo de acesso ao jogo.

O Download Manager trabalha com uma fila de prioridades. Os downloads de maior prioridade são os Forçados, em seguida os por Demanda e por último os Contínuos. Observe que os downloads por demanda só entram na fila quando o jogador chega à viewport a qual ele pertence. Nos downloads contínuos, a ordenação de download é feita baseada no vetor distância entre o MovieClip que será baixado e a região corrente de exibição. Essa ordenação é dinâmica, de forma que se o jogador for para outra posição na cena, primeiro é baixado o que já está em curso e é recalculada a prioridade. Esse recálculo é feito toda vez que há uma ativação/desativação de viewport. O usuário pode pausar o sistema de download e reiniciá-lo se quiser.

4.3. Scene Manager

O usuário pode precisar especificar mais de uma cena, para isso, foi criado o Scene Manager, que permite o gerenciamento de várias cenas pela Reflex.

Cada cena possui uma Tabela Hash própria e um Download Manager próprio. O Scene Manager permite que uma cena seja pausada, ou parada e que outra cena possa ser exibida, funcionando assim como uma controladora de fluxo de cenas. Dessa forma, é possível criar jogos que possuam túneis escondidos, ou mesmo mostrar as opções ou créditos durante uma cena.

4.4. Physics

Durante a especificação do cenário, o usuário pode escolher os métodos de tratamento de colisão de um movieclip, ou criar uma *Collision Zone*.

Uma Collision Zone é uma região lógica em que há um tratamento de colisão, contudo, não há um objeto real naquela região. A utilidade dessa estrutura pode ser percebida quando se usa jogos de plataforma, assim sendo, basta existir um desenho na imagem da cena com as plataformas e então pode-se criar uma Collision Zone na região do desenho da plataforma. A Figura 04 ilustra essa situação.

Figura 04: *Plataformas como Collision Zone do jogo Ice World com a Reflex*

Observe que os quadros apontados são estruturas lógicas.

Quando se usa um movieclip, as opções de colisão podem ser:

- Bounding Box
- Circular
- Bloco

- Sem Colisão

A Colisão Bounding-Box calcula a colisão assumindo as dimensões do movieclip. A colisão de bloco é similar a anterior, mas o usuário é quem determina as dimensões a serem consideradas, bem como o pivô de referência. Quando o usuário especifica a opção sem colisão, o motor de Física simplesmente ignora o movieclip.

O Sistema de Física não realiza as transformações sobre os movieclips devido às colisões: na verdade ele apenas informa a ocorrência de uma colisão e fornece dados sobre a mesma, caso se queira que seja feito os cálculos, deve-se especificar essa informação no cenário ou utilizar os serviços do Collision Solver.

O motivo de permitir que o usuário apenas queira saber se houve colisão e não realizar as movimentações possui aplicações como, por exemplo, num jogo estilo Super Mario Bros, uma colisão na parte inferior de um determinado bloco gera uma animação e dá pontos para o jogador, porém, em um jogo de espaçonaves isso pode retirar vidas e reiniciar a fase.

Um detalhe importante é que a Física só é simulada nas viewports em exibição, apesar disso restringir algumas ações, no caso do Flash, que não possui um bom desempenho com cálculos de ponto flutuante, tal medida aperfeiçoa o jogo. O Sistema de Física da Reflex ainda não fornece muitos recursos, a parte de colisão poligonal ainda está em estudo, contudo, ela é de extrema importância, já que, pode-se usá-la para aproximar a colisão de imagens, usando algoritmos de processamento de imagem, por exemplo, aproximação (interpolação) poligonal de imagens binárias.

4.5. I/O System

A camada de I/O (IOS) fornece para o motor, formas de ler arquivos de configuração, arquivos de cenário, obter o estado do teclado e do mouse. A IOS fornece ainda métodos de salvamento de dados e serialização de informações para XML, isso é útil para criar sistema de salvamento de jogo.

5. Utilização da Reflex

Para a utilização da Reflex, deve ser criado uma instância, ou objeto, ReflexEngine. Essa instância é capaz de gerenciar, teoricamente, quantos cenários o usuário quiser.

Quando se cria essa instância, o usuário pode passar um arquivo de configuração para o motor, de forma que, dependendo do projeto, algumas funcionalidades podem ser desativadas para obter um melhor desempenho. A Figura 05 apresenta um exemplo de criação.

```
0  import Reflex;
1  var oReflex:ReflexEngine;
2
3  oReflex = new ReflexEngine();
4
5  // ou, passando configuração
6  oReflex = new ReflexEngine("conf.xml");
7  oReflex.loadScene("cena01.xml");
```

Figura 5: *Instanciação do Motor*

O arquivo de configuração é uma descrição em XML, das funcionalidades do motor, sendo possível configurar os vários subsistemas do motor.

Após a instanciação do motor, o usuário deve definir os cenários que a Reflex deve gerenciar. A definição dos cenários é feita por meio de arquivos XML, onde são explicitados todos os parâmetros do cenário, como posição, forma de colisão, Collision Zones e forma de carregamento de movieclips. Na Figura 06 é mostrado um exemplo desses arquivos.

```
0 <reflex version="0.7.1" ... >
1   <scene name="first" width="1024"
2     height="auto" xstart="200" ystart="400"
3     scrolltype="tile-seak-middle"
4     scrollseaktile="nave">
5
6       <tile type="movieclip" load="static"
7             name="nave" x="200" y="300">
8         <collision type="bounding-box">…
9       </tile>
10 ...
```

Figura 06: *XML de Especificação do Cenário*

No exemplo da Figura 06 define-se um cenário de largura de 1024 pixels e altura automática, ou seja, a engine irá atribuir como altura o valor da posição somado à altura do movieclip mais distante, essa funcionalidade é especialmente útil, pois, os cenários podem ser ampliados sem necessidade de mudar esses atributos da cena.

Além disso, especifica-se a posição inicial da cena, forma de scroll, que no caso, a tela irá acompanhar o movimento do movieclip "nave" e deixá-lo no centro da tela. Outras formas podem ser usadas. Pode-se usar, ainda, a opção *manual*, que faz com que o scroll só ocorra quando se chame explicitamente a função que o realiza. Existem muitas outras opções de cenário.

Quanto à especificação de movieclips, pode-se especificar o tipo do movieclip, real ou Collision Zone, a forma de carga (load), que permite fazer carregamento dinâmico forçado, por demanda, contínuo e carregamento estático, além de especificar se deve ser feito cálculo de colisão e a forma de colisão, bounding-box ou outras.

É permitido acrescentar movieclips durante a execução, através dos comandos de cenários e, na inclusão, ainda, pode-se especificar o tempo de vida do

movieclip, que é muito usado quando, por exemplo, há um disparo de uma arma.

Cada cena possui um ID e um nome, atributos usados para mudar o fluxo de exibição de cenas da Reflex, através de comandos como o `showScene("name")`. Dessa forma, o usuário pode solicitar que o motor pause uma cena e mostre outra.

Para desenhar uma cena, o usuário deve utilizar o comando `reDraw(void)`. O objetivo de tal comando é fornecer meios para que o usuário possa criar o *MainLoop*, fluxo de execução do jogo, conforme a necessidade do projeto.

5.1. Eventos

A Reflex trabalha com sistema de eventos. Um evento pode ser entendido com uma Regra, mais especificamente, uma regra de reação [WAG98]. Essa regra pode ser reduzidamente verbalizada da seguinte maneira:

"Quando acontecer ***X*** faça **Y**!"

Nesse exemplo, X é um acontecimento qualquer, que pode ser o pressionar de teclas do teclado, movimentação do mouse e até mesmo tratamento de exceções e colisão de tiles e Y é uma ação a ser tomada, na verdade, uma reação ao acontecimento X.

A Reflex oferece esse suporte por meio do registro de *funções de callback*. As funções de callback são as reações aos eventos, e do ponto de vista de programação, elas são as funções que serão chamadas quando um evento ocorrer. Essa metodologia é usada na maioria das aplicações, inclusive pela biblioteca GLUT do OpenGL [SHR05].

Os principais eventos que o motor disponibiliza para o usuário são os de Pressionar Teclas, Movimentação do Mouse, Detecção de colisão, Erros do sistema e informação do estado de download de movieclips.

Os dois primeiros tipos de eventos são muito comuns em todos os ambientes de programação, inclusive no Flash: a Reflex os disponibiliza para prover regularidade ao projeto de jogo. O Evento de detecção de colisão é um pouco mais complexo: para o entendimento do funcionamento de tal evento, considere-se o exemplo em que dois MovieClips colidam, quando a engine detecta que há uma colisão, ela chama a função de callback passando como parâmetros a referência para os objetos envolvidos na colisão. Gera-se uma chamada da seguinte forma:

```
oReflex.OnColide( Object1, Object 2);
```

Observe que a função testa a colisão elemento a elemento, infelizmente essa não é a melhor estratégia para detectar colisão, pois no caso de colisão múltipla isso pode levar a erros de programação. Para solucionar tal problema, foram criadas funções que retornam um vetor de colisões de um objeto específico como:

```
oReflex.RePhysics.getColisionList(Object1);
```

Dessa forma, o usuário pode saber todas as colisões que um determinado objeto sofreu e pode solicitar o serviço da Engine de Física para tratar essas colisões. Atualmente os recursos da Engine de Física estão ainda muito limitados, fornecendo apenas cálculos de colisão simples conforme já mencionado.

Outro tipo de evento muito útil é o de tratamento de erros, que fornece ao usuário informações de eventuais problemas de comunicação, falhas de download de movieclips e outros. O usuário pode ainda solicitar mais informações sobre o erro, bastando passar o identificador do erro, ErrorID. A chamada feita pela engine é do tipo:

```
oReflex.OnError( ErrorCode, ErrorID );
```

Por fim, um evento especialmente interessante é o de carregamento de movieclips. Nele são informados os progressos de download, quantidade de objetos na fila, velocidade estimada de download, tamanho total e outras informações pertinentes. Esses dados podem ser úteis para criação de um pre-loader[7] de uma fase. Se um evento não possuir função de callback, ele não é tratado e nem verificado pelo motor.

5.2. Uso Geral e GUI

Usando os recursos da engine, é possível criar os mais variados jogos, além de aplicações. Pode-se, por exemplo, desenvolver uma aplicação no estilo do famoso Google Maps e WikiMapia. A flexibilidade é uma das características que dá nome à Reflex.

Por fim, todo jogo possui uma interface gráfica com o usuário, GUI - ***G**raphical **U**ser **I**nterface*, que deve facilitar a inserção do jogador no jogo. Devido aos recursos do Flash, isso é muito simples de desenvolver, visto que o Flash dá suporte íntegro a vários recursos gráficos, tais como, botões, textos, caixa de edição entre outros.

6. Estudo de Caso: SPACE GAME

O Space Game é um jogo 2D cujo objetivo é atravessar um labirinto, achar o monstro chefe e destruí-lo, em um enredo clássico. Contudo, a nave não pode colidir com as paredes do labirinto e nem ser atingida pelos inimigos. A nave conta com 3 vidas e um tiro que rebate na parede não acerta a própria nave.

No jogo, para a nave movimentar-se, ela deve acionar os motores de propulsão, usando a tecla ↑, e a direção

[7] *Pre-loader pode ser entendido como uma tela que mostra o progresso de carregamento de alguma coisa*

de aceleração é definida pela rotação da nave, conforme o esquema da Figura 07:

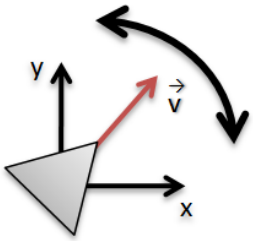


Figura 07: *Movimentação da Nave do jogo Space Game*

Para girar a nave usa-se as teclas →,←, sendo que → gira no sentido horário e ← no sentido anti-horário. Além disso, existe um atrito viscoso que reduz a velocidade da nave com o passar do tempo.

Há, também, dois tipos de inimigos na fase e um chefe. Os inimigos possuem movimento Vertical ou Horizontal, e o chefe movimenta-se circularmente e dispara na direção da nave. Os personagens do jogo podem ser vistos na Figura 08.

(a) (b) (c) (d)

Figura 08: *Personagens do Jogo Space Game:* ***(a)*** *Chefe* ***(b)*** *Inimigo,* ***(c)*** *Nave do Jogador e* ***(d)*** *Arma do Chefe*

O cenário é uma imagem de 2907 x 3306 pixels a qual poderia ter sido dividida, mas para simplificar o projeto sem a Reflex isso não foi feito. Foram utilizados retângulos e círculos como modelo de colisão nos movieclips. Na Figura 09 é mostrado o mapa em escala reduzida As regiões que não são exibidas no jogo foram cortadas da imagem.

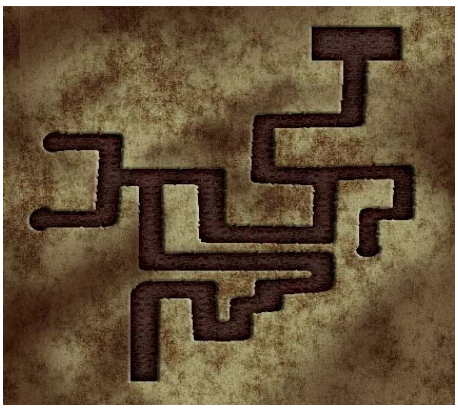


Figura 09: *Cenário do Jogo*

É importante observar que os dados detalhados, até agora, só dizem respeito ao funcionamento específico do jogo, de forma que, apenas a elaboração das regras do jogo e a modelagem do cenário foram definidas. Dessa forma, ainda não foi analisado como o sistema funcionará.

6.1. Versão sem Reflex

A versão sem o uso da engine mostrou-se demasiadamente complexa para ser desenvolvida, uma vez que, dado o tamanho do cenário, foi necessário muito cuidado para tratar as colisões com as paredes, lembrando que há side-scroll e a nave deve ficar no centro da tela, por isso o calculo da colisão tornou-se bastante complexo.

Contudo a programação dos personagens, movimentação e disparo foi simples. Um fato importante é que o side-scrolling usado é diferente do descrito na seção 3, pois usando aquele método, o jogo ficava extremamente lento. A técnica utilizada é armazenar em um vetor todos os movieclips e a cada vez varrer o vetor, verificando se o movieclip deve ou não ser mostrado.

6.2. Versão com Reflex

Usando a Reflex, bastou criar o arquivo XML do cenário, no qual foi colocada a nave em um ponto de partida, espalhados os inimigos pela cena, definidas as Collision Zones e se posicionada a imagem de fundo. Na sequência foi configurada a cena para seguir a nave e mantê-la no meio da tela. O início desse arquivo pode ser visto na Figura 06.

A parte de programação dos personagens ocorreu da mesma forma que a realizada para a versão sem a Reflex. Nesse caso, foram alterados os parâmetros a serem modificados, ou seja, trocando a propriedade de movieclip pelas propriedades do ReObject.

O uso da Reflex permitiu a mudança do jogo, como, por exemplo, a posição dos inimigos, sem a necessidade de reprogramar.

6.3. Análise Comparativa

Para ambas as versões foram feitos os cálculos de FPS. Na Figura 10 é mostrado um código em ActionScript similar ao usado no cálculo.

```
 0  // Frame 1
 1  var iFPS:Number = 0;
 2
 3  var iTimeID:Number = 0;
 4
 5  //showFPS insere dado no vetor e reinicia
 6  iTimeID = setInterval(showFPS,1000);
 7
 8  // Frame 2
 9  iteraJogo(); // Faz tudo no jogo
10  iFPS++;
11  gotoAndPlay(2);
```

Figura 10: *Código de Cálculo de FPS*

Para comparar os resultados, o cenário foi subdividido em setores de análise, de forma que, sabendo que alguns setores apresentam mais movieclips que outros,

fosse possível analisar o desempenho dinâmico das técnicas usadas.

A Figura 11 mostra as seções analisadas e o desempenho médio na seção das duas versões do jogo.

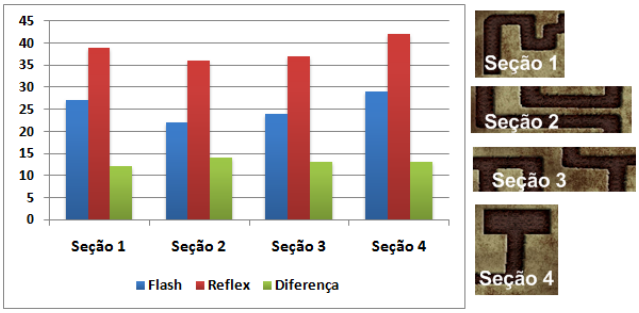


Figura 11: *Gráfico da Analise Comparativa de Desempenho*

O gráfico mostra que em seções de poucos movieclips, ambas apresentam bom desempenho, mas nas transições e regiões com mais movieclips o desempenho da versão **sem Reflex** caiu bastante, enquanto o da Reflex manteve-se estável e bem superior. Os benchmarks serão divulgados juntamente com a Reflex. O cálculo de erro foi feito via desvio padrão. Diante dos dados não foi possível fazer uma análise de testes de hipóteses usando uma distribuição estatística como t-Student, contudo percebe-se que há uma significativa melhora de desempenho com o uso da Reflex.

Utilizando a teoria de análise de algoritmos, percebe-se que parte da melhoria do desempenho é devido à redução da ordem do algoritmo proporcionado pelo uso da tabela hash.

7. Considerações Finais

A Reflex, conforme a Figura 06, encontra-se na versão 0.7.1 e está em fase de teste e validação de serviços, contudo, é possível notar que os benefícios de utilização, de fato, auxiliam na produção de jogos e aplicações, aumentam a performance e melhoram a organização dos recursos de um jogo.

Mesmo a Engine de Física ainda apresentar limitações a serem corrigidas, é possível com os atuais recursos criar vários jogos.

Por fim, a Reflex não deve ser vista apenas como um motor de jogos, mas sim como uma provedora de recursos, que podem auxiliar a realização de vários projetos. Além disso, é importante observar que os conceitos do desenvolvimento da Reflex podem ser estendidos para quaisquer outras plataformas, não ficando restrito apenas ao Flash.

8. Trabalhos Futuros

Um serviço que a Reflex ainda não possui, mas que está em processo de desenvolvimento é o de fornecer métodos para jogos multiplayer através do Flash Communication Server. Tal funcionalidade permitirá que a produção de jogos multiplayer para Flash seja muito mais facilitada, para isso estão sendo elaborados protocolos para comunicação de dados da Reflex.

Outra linha de trabalho é a produção de ferramentas que auxiliem o processo de criação de arquivos de configuração da Reflex e dos cenários. Devido ao fato do formato usado ser XML, que é totalmente aberto, esse processo torna-se simples e pode ser realizado por qualquer grupo de pesquisa, contudo o desenvolvimento dessa ferramenta é indispensável.

Além desse fato, muito pode ser acrescentado nos serviços de Física, e a equipe do ModeLab está estudando formas de integrar Engrenagens de Física de Flash, como a APE, para servir como provedora de recursos de Física e, dessa forma, permitir simulações mais complexas.

Atualmente, pretende-se criar uma versão para a Plataforma Microsoft XNA, de forma a possibilitar a criação de jogos para o Xbox 360.

Finalizando, o Projeto Reflex encontra-se, ainda, em processo de desenvolvimento que permitirá a inclusão de uma série de serviços e facilidades, tais como, PathFind, visão isométrica, entre outros. Dessa forma, em um futuro próximo, espera-se pode disponibilizar uma versão dessa ferramenta em um estágio mais avançado para que possa ser utilizada e testada por grupos de pesquisa e interessados na área de desenvolvimento de jogos.

9. Agradecimentos

Este estudo é parcialmente suportado pelo FINEP, CNPq, CAPES e FACITEC – Conselho Municipal de Pesquisa em Ciência e Tecnologia de Vitória, ES – Brasil.

Agradecimentos especiais aos Artistas Leonardo Rangel, Leonardo Silva Vago e a Tiago Pinheiro Teixeira pela produção das imagens usadas nos jogos.

10. Referências

[ADO06] Adobe Systems Incorporated. (2006). *Flash 8 Documentation*. Retrieved 04 04, 2007, from Adobe Systems Incorporated: http://livedocs.adobe.com/flash/8/main/

[BOO94] BOOCH, G. (1994). *Object-Oriented Analysis and Design with Applications.* Addison-Wesley.

[COM04] COMBS, N., & ARDOINT, J.-L. (2004). Declarative versus Imperative Paradigms in Games AI1. *AAAI 2004 Workshop on Challenges in Game AI* .

[COR02] CORMEN, T. H., LEISERSON, C. E., RIVEST, R. L., & STEIN, C. (2002). *Algoritmos: Teoria e Prática.* Rio de Janeiro: Campus, Elservier.

[COS05] COSTA, S., BELFORT, R., & ARAÚJO, A. (2005). *A Indústria de Desenvolvimento de Jogos Eletrônicos no Brasil*. Retrieved Maio 20, 2007, from Abragames: http://www.abragames.org/docs/PesquisaAbragames.pdf

[CRA03] CRAWFORD, C. (2003). *Chris Crawford on Game Design.* EUA: New Riders Publishing.

[DAY83] DAY, J. D., & ZIMMERMANN, H. (1983, Dezembro 12). The OSI Reference Model. *Proceedings of the IEEE* , pp. 1334-1340.

[ECM99] ECMA International. (1999). *Standard ECMA-262 - ECMAScript Language Specification.* Retrieved Maio 07, 2007, from ECMA International: http://www.ecma-international.org/publications/files/ecma-st/ECMA-262.pdf

[FEI06] FEIJÓ, B., PAGLIOSA, P. A., & CLUA, E. W. (2006). Visualização, Simulação e Games. In K. BREITMAN, & R. ANIDO, *Atualizações em Informática* (pp. 127-186). Rio de Janeiro: Editora PUC-Rio.

[FRA99] FRATERNALI, P. (1999). Tools and Approaches for Developing Data-Intensive Web Applications: A Survey. *ACM Computing Surveys* , 227-263.

[GOS96] GOSLING, J., JOY, B., & STEELE, G. L. (1996). *The Java Language Specification.* Addison Wesley Publishing Company.

[MAG05] MAGALHÃES, A. (2005). *O que esperar da internet nos próximos anos?* Retrieved 05 10, 2007, from IBOPE Inteligência, IBOPE NetRatings, Internet, Notícias 2006: http://www.ibope.com.br (Pesquisas » Internet » Notícias)

[PRE05] PRESSMAN, R. S. (2005). *Software Engineering - A Practitioner's Approach.* The McGraw-Hill Companies, Inc.

[RIB05] RIBEIRO, F. (2005). *Design de Jogos: O Design sob a ótica da interatividade e do desenvolvimento de projeto.* Florianópolis.

[SAL04] SALEM, K., & ZIMMERMAN, E. (2004). *Rules of play – Game Design fundamentals.* Cambridge, Massachusetts: The MIT Press.

[SAN06] SANTOS, R. J., & BATTAIOLA, A. (2006). Análise de Tecnologias para a Implementação de Jogos Web. *Digital Proceedings of the V Brazilian Symposium on Computer Games and Digital Entertainment.*

[SHR05] SHREINER, D., WOO, M., NEIDER, J., & DAVIS, T. (2005). *The OpenGL Programming Guide - OpenGL Programming Guide.* Addison-Wesley Professional.

[SIL06] SILVA, R. M., FERRACIOLI, L., & CAMARGO, F. S. (2006). Alerta Verde : Um adventure educacional. *Digital Proceedings of the V Brazilian Symposium on Computer Games and Digital Entertainment,.*

[WAG98] WAGNER, G. (1998). *Foundations of Knowledge Systems with Applications to Databases and Agents.* Dordrecht: Kluwer Academic Publishers.

[WOR05] World Wide Web Consortium(W3C). (2005). *Document Object Model (DOM).* Retrieved Maio 02, 2007, from W3C - The Architecture Domain: http://www.w3.org/DOM/

[WOR06] World Wide Web Consortium(W3C). (2006). *The Extensible Stylesheet Language Family (XSL).* Retrieved Maio 02, 2007, from W3C - The Architecture Domain: http://www.w3.org/Style/XSL/

Avaliando a Usabilidade de um Jogo através de sua Jogabilidade, Interface e Mecânica

Carlos Teixeira Karen Daldon Omar Buede Milene Silveira

Pontifícia Universidade Católica do Rio Grande do Sul, Faculdade de Informática, Brasil

Resumo

O sucesso de um jogo depende de sua apreciação positiva pelo usuário. Ou seja, se ele não tiver qualidades de uso (seja uma boa interface ou um bom desafio, por exemplo) provavelmente o usuário procurará uma outra alternativa de entretenimento. Para garantir esta qualidade de uso, é fundamental que os testes de software realizados durante o desenvolvimento de um jogo incluam testes de usabilidade, os quais representam a perspectiva do usuário. Neste sentido, este artigo apresenta uma proposta de teste de qualidade de interação específica para a área de jogos, na qual a usabilidade de um jogo é avaliada a partir de sua jogabilidade, interface e mecânica.

Palavras-chave: testes de usabilidade, jogabilidade, interface, mecânica

Contato dos Autores:
carlos.teixeira@pucrs.br
kdaldon@uol.com.br
omar.buede@gmail.com
milene.silveira@pucrs.br

1. Introdução

Usabilidade é um atributo de qualidade de software que permite avaliar a qualidade de interação de um sistema, de acordo com fatores que os projetistas definem como sendo prioritários ao sistema, dentre eles [Nielsen 1993; Preece 1994]:

- Facilidade de aprendizado (quão fácil é para um usuário aprender a usar o sistema);
- Facilidade de uso (quão fácil é, para um usuário infreqüente, recordar o que precisa para usar a interface);
- Eficiência de uso (uma vez aprendido a usar uma interface, quão rápido um usuário consegue executar tarefas através da mesma);
- Produtividade (se as tarefas são realizadas melhor – e mais rapidamente – através deste sistema);
- Satisfação do usuário (quão agradável é seu uso para o usuário);
- Flexibilidade (se existem caminhos diferentes para realizar uma mesma tarefa ou se existem usos diferentes para uma mesma aplicação);
- Utilidade (se além de funcionar, a aplicação é adequada às necessidades do usuário);
- Segurança no uso (se existe proteção contra erros e quão fácil o usuário consegue recuperar-se destes).

Jogos eletrônicos, mais do que qualquer outro tipo de software, dependem muito deste atributo (usabilidade). Um software de uso comercial, necessário para o usuário efetuar seu trabalho, por exemplo, pode não ser muito eficiente ou não ser do "gosto" do usuário, mas este necessita utilizá-lo a fim de realizar suas tarefas cotidianas; não há outra opção. Um jogo, por outro lado, se não tiver qualidades de uso, pode levar o usuário a procurar uma alternativa que melhor o satisfaça.

Para que isso não aconteça, os projetistas devem seguir uma série de critérios necessários para que o jogo cumpra sua função e seja bem assimilado pelo usuário. Detectar falhas ou erros de projeto e desenvolvimento nos três pilares que constituem a usabilidade em um jogo (interface, jogabilidade e mecânica), antes que este seja finalizado e lançado, é imprescindível. Esta necessidade (de cuidado com a usabilidade em jogos) é tamanha, que é possível encontrar, por exemplo, publicações especializadas da área de Interação Humano-Computador (IHC), com edições específicas para tratamento do tema [IwC 2007].

Além disto, neste contexto, vê-se, também, o crescimento continuo da indústria de jogos eletrônicos no Brasil e, em paralelo, a falta de serviços e mão de obra especializados nesta área (testes de jogos). Assim, surge a necessidade de se desenvolver um processo de testes sólido para a qualificação deste tipo de software, testes estes que incluam a perspectiva do usuário, pois é ele quem definirá o grau de sucesso de um jogo.

Assim, este artigo visa apresentar uma proposta de teste de qualidade de interação específica para a área de jogos[1], na qual a usabilidade de um jogo é avaliada a partir de sua jogabilidade, interface e mecânica. Neste sentido, serão apresentados os fundamentos que embasam esta proposta, bem como o relacionamento da mesma com o que há na literatura (cabe destacar que o assunto será tratado – neste artigo – em maior profundidade no que tange a jogabilidade e interface de um jogo). Além disto, será feita uma discussão inicial sobre as possibilidades de inserção deste tipo de

[1] Este trabalho foi desenvolvido em colaboração com a empresa **Zero-Defect Test House.**

avaliação durante o processo de projeto e desenvolvimento de jogos eletrônicos, bem como sobre perspectivas de trabalhos futuros.

2. Trabalhos Relacionados

Na área de IHC, dois dos tipos de avaliação de interfaces/qualidade de interação mais utilizados são os métodos de avaliação por inspeção e os de avaliação através de testes com usuários.

Na avaliação por inspeção [Mack e Nielsen 1994], dependendo do método utilizado, um ou mais especialistas na área de IHC e/ou na área de domínio da aplicação sendo avaliada analisam a interface, inspecionando-a a fim de verificar se ela encontra-se de acordo com critérios de usabilidade pré-definidos. Dentre os tipos mais utilizados de avaliação por inspeção, encontram-se a avaliação heurística [Nielsen 1993 e 1994], o uso de listas de verificação (*checklists*) [Dias 2003; Ergolist 2006] e o percurso cognitivo [Wharton et al. 1994].

Já na avaliação através de testes com usuários [Rubin 1994], serão estes, os usuários finais da aplicação em questão, que irão avaliar a interface. A equipe responsável pelos testes faz o planejamento do mesmo, incluindo desde o local onde a avaliação vai ser realizada, os equipamentos, recursos humanos necessários e tarefas a serem realizadas até o método a ser utilizado para captura e análise dos dados. O mais conhecido destes testes são os testes de usabilidade [Rubin 1994], sendo que, recentemente, também vem crescendo o uso de testes de comunicabilidade [Prates et al. 2000]. Ambos se diferenciam, principalmente, na forma de analisar os dados obtidos durante as avaliações.

Já nos estudos efetuados sobre avaliações voltadas a esta área, uma classificação interessante de itens da usabilidade de um jogo é proposta em [Clanton 1998]. Este autor sugere três principais itens de usabilidade a serem avaliados em um jogo:

- Jogabilidade: relativo aos objetivos do jogo, como descobri-los e superá-los;
- Interface: relativo ao nível de percepção (visual e sonora) e atividade motora;
- Mecânica: relativo à "física" do jogo.

Além deste, outro método interessante encontrado é o RITE (*Rapid Iterative Testing and Evaluation*), um método de avaliação de usabilidade desenvolvido pela Microsoft [Medlock et al. 2002]. Este é um teste prático, no qual jogadores são convidados a utilizar determinado jogo, dentro de um ambiente controlado e monitorado por pessoas qualificadas, a fim de detectar problemas ainda no estágio de produção do jogo.

A vantagem deste método é já possuir dados a respeito dos jogos, jogadores e das plataformas, catalogados e definidos de forma a serem utilizáveis em qualquer caso, não havendo assim a necessidade de iniciar uma nova pesquisa a cada jogo, sendo necessário apenas um analista de testes com conhecimento sobre o método. Entretanto, seu problema principal é que ele depende não só de uma equipe qualificada para planejamento e acompanhamento dos testes, mas, também, de um determinado número de participantes, gerando assim custos elevados e a necessidade de um espaço físico dedicado, além de não abordar de maneira específica e organizada os três principais itens da usabilidade vistos anteriormente [Clanton 1998].

3. Jogabilidade, Interface e Mecânica

Embora as categorias de Clanton [Clanton 1998] tenham definição clara e suficiente para entender como funcionam os jogos do ponto de vista de um leigo, não são suficientes para um completo teste de qualidade de interação, que demanda mais definições para ser executado de maneira eficiente. O ideal, no caso de um teste, é dar maior detalhe e definição para cada item das categorias propostas pelo autor, adicionando critérios de inspeção a cada um destes itens, a fim de auxiliar na divisão de etapas e abrangência.

A fim de detalhar cada uma destas etapas, foram analisadas as heurísticas de Nielsen [Nielsen 1994]. Estas heurísticas são tradicionalmente usadas para a avaliação de usabilidade de sistemas interativos, então foi feito um estudo inicial a fim de usá-las como base do processo de avaliação proposto. Nesta análise, foram encontrados alguns problemas, dentre os quais são ressaltadas a ambigüidade em determinados itens, quando aplicados a jogos, o que torna o teste extremamente dependente da interpretação do testador, e a falta de especificidade da avaliação, na qual aspectos importantes de um jogo não são avaliados.

Federoff [Federoff 2002] sugere adaptar as heurísticas de Nielsen às três categorias de Clanton. Todavia, a partir das sugestões desta autora, decidiu-se não apenas adaptar, mas, também, organizar de maneira estruturada os itens de avaliação, na forma de uma *checklist*.

Em IHC, com o uso de *checklists*, verifica-se a conformidade da interface a uma série de critérios disponíveis em uma lista de diretrizes (*checklists*). Neste tipo de método, ao contrário da avaliação heurística, são as qualidades das diretrizes e não dos avaliadores que determinam o sucesso da avaliação. Sua grande vantagem é permitir a avaliação da interação por um profissional que não seja especialista em usabilidade, visto que o mesmo deve inspecionar a interface de acordo com critérios bastante específicos. Esta vantagem leva a um rigor muito grande na criação das diretrizes a serem verificadas.

As *checklists* desenvolvidas nesta pesquisa foram criadas de acordo com as três categorias anteriormente citadas (Jogabilidade, Interface e Mecânica) e por profissionais das diferentes áreas relacionadas (Psicologia, *Design* e Computação, respectivamente).

Como o detalhamento de cada item das *checklists* não pode ser divulgado por motivos de confidencialidade, o foco deste trabalho está na discussão de suas categorias de base: jogabilidade, interface e mecânica.

3.1 Jogabilidade

Jogabilidade é o item responsável pelo conceito do jogo, que avalia sua proposta e os estímulos psicológicos por ele provocados. Para tal, é importante o conhecimento de questões acerca do público-alvo (faixa-etária, sexo, escolaridade, línguas estrangeiras de domínio), da plataforma utilizada (consoles, computadores, arcades) e do desafio (estímulos psicológicos). Além disso, é necessário que o objetivo central do jogo esteja definido e especificado adequadamente.

A seguir, serão discutidos aspectos da jogabilidade referentes à memória, diversão e aprendizagem, desafios e habilidades, reforço e punição, preferências e sociabilidade.

3.1.1 Memória

O cérebro humano, responsável pela coordenação do funcionamento do corpo, possui uma estrutura central para os processos de pensamento. Esta é denominada memória, que é o registro da experiência que é subjacente à aprendizagem. É a capacidade mental de reter, recuperar, armazenar e evocar informações [Anderson 2005]. Segundo Sternberg (2000), o modelo tradicional de memória, chamado de Modelo dos Três Armazenamentos, divide a memória em: sensorial, que estoca informações por períodos muito breves; de curto prazo, que armazena informações por períodos mais longos; e, por fim, de longo prazo, que possui uma capacidade muito grande e estoca informações durante períodos longos. Durante o jogo, o jogador utiliza a memória de longo prazo quando associa as informações do software com aquilo que já conhece. A memória sensorial é utilizada todo o tempo, já que registra os estímulos do jogo. E a memória de curto prazo é usada para novos registros que são revistos a todo o momento no jogo, como regras, enredo, entre outros.

Na memória, ocorre um processo denominado *chunking*, uma estratégia de uso de informações memorizadas na memória de curto-prazo. Consiste em agrupar algumas informações e tratá-las como "unidades" autônomas. A memória de curto prazo poderia abrigar somente 5 a 9 *chunks* (pedaços) de informação. Este processo reduz a complexidade da informação. Todavia, uma estrutura hierárquica vai progressivamente crescendo e nos permitindo aprender outras estruturas, cada vez mais complexas e sofisticadas [Koster 2005].

A memória é, a todo o momento, associada a novos estímulos. Quando o jogo oferece informações que são relacionadas ao que já se tem registrado na memória, o usuário se tranqüiliza, pois as informações são familiares e, portanto, mais facilmente passíveis de entendimento. Personagens e ações que ocorrem em tempo real facilitam esse processo de identificação. Geralmente, existe uma resposta rápida ao estímulo familiarizado, ao contrário do estímulo desconhecido. Quando aparecem no jogo informações difíceis de serem associadas à memória, a ansiedade do jogador aumenta, exigindo mais esforço intelectual para emitir uma resposta àquele estímulo. Dessa forma, o jogo deve possuir um equilíbrio, composto de uma parte conhecida, que seja familiarizada para o jogador, e de outra nova, com possibilidades diferentes.

É fundamental que os aspectos individuais dos jogadores sejam considerados, assim como as diferentes reações possíveis que podem ocorrer frente aos estímulos transmitidos pelo jogo. Para isso, o jogo eletrônico possuir a capacidade de o jogador escolher, ao menos em uma parte, características do personagem e do desafio, facilita o encontro de informações conhecidas e, portanto, uma melhor adaptação do usuário ao jogo. Um exemplo é o jogo *The Sims*, no qual o jogador monta sua casa de acordo com suas preferências, a partir de uma série de opções (Figura 1).

Figura 1: *The Sims* (Eletronic Arts)

Algumas possibilidades já são conhecidas como atrativas para a grande maioria dos jogadores. O exercício de poder e heroísmo costuma provocar muitas emoções e interesses. Outras questões como o prazer, a capacidade da fantasia parecer real, as descobertas do jogo, a produção da história e a identificação com ela, o curso de obstáculos e a possibilidade de interagir com outros jogadores são exemplos de características que atraem os mais diversos tipos de jogadores.

Outro processo utilizado pela memória é o *grok*, que significa que a pessoa entendeu algo tão completamente que se torna inteiro, unificado com aquilo. É um profundo entendimento [Koster 2005]. Tal conceito é essencial para demonstrar a importância das regras e de um enredo consistente, de forma que o jogo seja um fluxo de acontecimentos que possui uma essência de organização única. As mudanças a cada fase são atrativas, mas não devem ultrapassar os objetivos e regras do jogo em demasiado, para que o usuário entenda o jogo de forma global e consiga conectar-se a ele completamente. Caso contrário, o jogador acaba rejeitando o jogo.

3.1.2 Diversão e Aprendizagem

Conhecer os processos da memória facilita o entendimento, pelo menos em parte, do funcionamento psicológico dos jogadores. Todavia, na realidade, o que motiva realmente uma pessoa a jogar é a diversão, o prazer que aquele momento produz. Diversão é tudo que o nosso cérebro sente prazer, a liberação de endorfina no organismo. A diversão nos jogos é o ato de resolver enigmas e, nesse contexto, a aprendizagem é o processo cognitivo principal [Koster 2005]. As mudanças nos jogos eletrônicos são necessárias, porém devem ocorrer de forma progressiva, para que o usuário vá inserindo nos seus registros novos modelos de informação antes desconhecidos. É importante que o jogador tenha certo controle do jogo, de suas ferramentas e fluxo. Entretanto, à medida que o jogo prossegue, são exigidas do jogador decisões mais difíceis e complexas, evitando que o jogo fique monótono. Essas mudanças e novos modelos garantem um aprendizado contínuo, que é traduzido como diversão para o usuário. Neste contexto, inserem-se não só as trocas de desafio, mas também as mudanças de cenário e estrutura do jogo, que acabam por participar do processo de fascinação do jogador pelo software. No decorrer do jogo *Tomb Raider II Dagger of Xian*, os cenários se modificam e, algumas vezes, inclusive o transporte dos personagens são alternados, promovendo inovações atrativas ao jogador (Figura 2).

Figura 2: *Tomb Raider II: The Dagger of Xian* (Core Design / Eidos Interactive)

Divertir-se é aprender, e aprendizagem é o processo pelo qual modificações duradouras ocorrem no potencial comportamental como resultado da experiência. O jogo deve possuir a capacidade de estar sempre ensinando novas informações, em uma quantidade ideal, que não gere desconforto ou insatisfação.

3.1.3 Desafios e Habilidades

A mudança de desafio está relacionada aos tipos de habilidades que, no decorrer do jogo, vão sendo exigidas do jogador. Habilidade é a capacidade do ser humano de realizar algo [Anderson 2005]. Segundo Koster (2005), a verdadeira diversão vem de desafios que estão sempre no limite de nossas habilidades. As pessoas possuem diferentes inteligências, e escolhem seus jogos de acordo com o problema que acreditam possuir mais habilidades para resolver. Segundo Armstrong (2001), Howard Gardner criou a teoria das inteligências múltiplas, na qual agrupa as capacidades humanas em oito categorias (inteligências) abrangentes. São elas: lógico-matemática, lingüística, espacial, musical, corporal-cinestésica, intrapessoal, interpessoal e naturalista. E, cada ser humano, tem maiores habilidades em uma (ou mais) destas áreas.

As habilidades exigidas em um jogo devem ser organizadas de modo que sejam promovidas novas formas de aprendizado ao usuário. Geralmente, o jogador escolhe jogos eletrônicos que, inicialmente, exijam habilidades já desenvolvidas em si próprio. É importante o cuidado na produção do jogo para que nem habilidades demais, nem de menos, sejam solicitadas ao jogador. Devem ser considerados diversos aspectos, como idade, nível de escolaridade e outras características do público-alvo que são influentes na aquisição de habilidades.

3.1.4 Reforço e Punição

O processo de reforço e punição aumenta o controle sobre o comportamento dos jogadores, e os resultados ou estímulos que são estabelecidos podem ser desejáveis (recompensas) ou aversivos (punições). A recompensa é um dos componentes-chave do sucesso de um jogo, pois se não existe vantagem quantificável em fazer algo, o cérebro provavelmente o descartará imediatamente, considerando o jogo irrelevante [Anderson 2005]. O reforço promove o comportamento, através da apresentação de uma recompensa ou retirada de um estímulo indesejável. Já a punição diminui a freqüência do comportamento, através da apresentação de um estímulo aversivo ou pela retirada de algo desejável. O sistema de reforço e punição é uma ótima ferramenta que auxilia o produtor de jogos a comunicar ao jogador as regras do jogo. Tal ferramenta deve gerar motivação ao usuário, para que este queira continuar seguindo em frente e conquistando novos objetivos. O jogo eletrônico deve possuir várias possibilidades de *feedback*. Essas respostas devem ser consideradas a nível de ação,

como qualquer ato do jogo, e também a nível global, no qual o jogador possa receber informações sobre seu desempenho em toda a fase ou em todo o jogo, comparando-se a outros jogadores.

3.1.5 Preferências

Existem, também, outros fatores que devem ser considerados na avaliação de um jogo, como determinadas preferências que não possuem explicações específicas, remetendo à questão da satisfação subjetiva do jogador. As diferenças de gênero estão inseridas neste contexto.

Os sexos masculino e feminino possuem diferenças no processamento das informações, além de preferências próprias, geralmente aprendidas culturalmente. As mulheres querem jogos com sistemas abstratos simples, com menos raciocínio espacial e maior ênfase nas relações interpessoais, narrativa e empatia. Os homens preferem jogos que enfatizam o exercício do poder e o controle do território [Koster 2005]. Os dados desta categoria foram resultados de diversas pesquisas que chegaram a tais conclusões de preferências masculinas e femininas. São dados importantes, que podem direcionar para qual público o jogo será mais atrativo. Entretanto, cada dia mais, os estilos e as preferências dos sexos estão integrados, diminuindo as diferenças entre homem e mulher neste contexto.

3.1.6 Sociabilidade

Um movimento entre os jogadores de jogos eletrônicos vem sendo percebido no decorrer dos anos. Enquanto há um tempo era valorizado o jogo individual, atualmente o ato de jogar está se transformando, cada vez mais, em uma atividade social [Jones 2004]. A possibilidade de jogar com outros parceiros e adversários permite criar novas estratégias e intercambiar soluções para os desafios, tanto nos jogos que envolvem apenas dois jogadores por jogada, como naqueles que podem ser jogados com múltiplos usuários, nos quais as trocas podem acontecer em um mesmo espaço geográfico ou na rede [Alves 2005].

É importante que o jogador tenha a possibilidade de jogar com outros jogadores, seja através de um jogo cooperativo ou competitivo. Todavia, deve-se atentar para as características do jogo neste contexto, que deve possuir, por exemplo, diferentes descobertas e *feedback*s para jogadores com experiências distintas. Outra questão a considerar é que o jogo, como meio de expressão social, precisa favorecer a interação entre jogadores. Ferramentas como forma de expressão e comunicação de emoções e de aspectos pessoais, decisões de um jogador que afetam as decisões do outro jogador, entre outras, são estratégias importantes em um jogo para que haja o reconhecimento dos jogadores perante os demais, favorecendo a relação. No jogo *Sid Meier's Civilization III*, há a possibilidade de escolher as mais variadas características dos personagens, para que o jogador identifique-se e demonstre sua identidade aos demais jogadores (Figura 3). Neste contexto, o jogo insere-se como um instrumento de interação social, abrindo caminhos diferentes para a produção de produtos que enfatizem tal característica.

Figura 3: *Sid Meier's Civilization III* (Firaxis Games)

A jogabilidade de um jogo eletrônico é essencial para que este seja atrativo e promova a diversão do jogador. Avaliar a interação do usuário com o jogo, neste aspecto, auxilia na elaboração de um produto mais assertivo ao mercado consumidor. A base do ser humano são seus processos psicológicos e, dessa forma, é muito importante que os jogos eletrônicos estejam adequados a tais funcionamentos, para que a receptividade do jogador ao jogo seja maior e mais profunda.

3.2 Interface

A interface é a parte responsável pela comunicação direta do software com o usuário, e indireta dos desenvolvedores com o mesmo. Ela é o elo que aproxima e une o jogador ao jogo [Macedo Filho 2005] e que, juntamente com jogabilidade e mecânica, faz com que este se torne um grande atrativo ou não. Ela pode ser dividida em três partes: visual, sonora e sensível, segundo os sentidos humanos utilizados na percepção dos jogos eletrônicos. O sentido mais utilizado pelos jogadores é a visão. Com ela, percebem-se os diversos elementos visuais que nos proporcionam obter as informações relevantes para o entendimento de um jogo. Caso estes elementos não sejam adequadamente percebidos, todo o objetivo lógico e de desenvolvimento do jogo podem acabar não sendo atingidos e, assim, o usuário não se diverte e não aprende. Além de considerar o jogo muito ruim, também não consome mais e ainda aconselha os demais jogadores a não consumirem.

3.2.1 Parte Visual

O primeiro critério a ser avaliado em uma interface é a estética. A partir dela é que todo o sentido do jogo irá ser exposto e transmitido ao usuário. Quando se fala em estética, não está se falando apenas em beleza e

criatividade, mas na adequação e organização dos textos, informações, imagens, cenários e personagens ao tema do jogo. Para se avaliar as condições estéticas de uma interface gráfica em um jogo, podem se utilizar os mesmos critérios usados na criação de peças gráficas como jornais, anúncios, revistas, etc. Estes critérios são explicados pela Gestalt, que é um conjunto de leis que padronizam certas tendências de como organizar informações de forma que sejam transmitidas de maneira mais clara, rápida e eficaz. Tais leis são baseadas no processo de percepção do cérebro. Segundo elas, o que acontece no cérebro não é igual ao que acontece na retina. A excitação cerebral não se dá em pontos isolados, mas por extensão. Nossa percepção é resultado de uma sensação global, na qual as partes são inseparáveis do todo [Gomes Filho 2000].

Neste contexto, inicia-se pela análise dos quesitos básicos necessários para que uma informação seja assimilada. O contraste com o fundo é necessário antes de tudo, para que se consiga identificar facilmente o que é informação e o que é ambiente. As informações devem ser possíveis de se enxergar, sem requerer um esforço maior por parte do usuário, visto que este tem que se concentrar em outros fatores além destes. Conforme se observa na figura 4, a falta de contraste, presente na informação "220", com a nuvem dificulta a leitura e acrescenta uma perda de tempo desnecessária para interpretação.

Figura 4: *Alex Kidd* (Sega)

Informações que são de difícil visualização podem comprometer o resto do desempenho do usuário no decorrer do jogo. Do mesmo modo, elas devem ser expostas de forma simples e objetiva, além de manterem uma hierarquia (cuidando, também, para que informações adicionais não estejam sobrepostas a outras mais relevantes) para que o raciocínio do jogador flua naturalmente e, assim, possa interpretá-las de forma mais rápida e, consequentemente, ter mais tempo para o pensamento lógico do jogo.

A partir de então, deve-se analisar a forma como as informações estão organizadas. Segundo as leis da Gestalt, como cita Gomes Filho (2000), os elementos semelhantes devem ser colocados próximos, pois tendem a serem vistos juntos e, conseqüentemente, constituírem unidades dentro do todo. Estas unidades são elementos únicos que constituem a forma, e devem ser agrupadas de tal forma que estabeleçam um fechamento visual, construindo um grupo total e completo. Nesses grupos que formam a interface como um todo é necessário haver uma harmonia, equilíbrio e ordenação visual para que as informações estejam organizadas e sejam interpretadas o mais rápido possível. Também é indispensável uma coerência de linguagem e estilo formal das partes para que estas fiquem de acordo e relacionadas entre si e, assim, formem uma unidade geral. A interface como um todo deve estar bem equilibrada, aproveitando adequadamente a tela. O usuário não deve se fixar unicamente em um ponto, mas sim se concentrar em um todo, para sempre haver um dinamismo e uma continuidade no fluxo de informações. Monotonia e desorganização são alguns fatores que tornam um jogo ruim.

Muitas vezes, ao invés de escrever, utilizar signos, símbolos ou ícones para representar algo é muito mais dinâmico e prático para o usuário. Segundo Coelho Neto (1999), signo é aquilo que representa algo para alguém; símbolo é um signo que se refere ao objeto denotado através de uma associação de idéias produzida por uma convenção; e ícone é um signo que tem alguma semelhança com o objeto representado. Portando, deve-se utilizá-los desde que estes sejam de fácil interpretação e expressem, para o usuário, seu devido significado.

Figura 5: *Grand Theft Auto: Vice City* (Rockstar North)

Como pode ser percebido na figura 5, os símbolos facilitam e agilizam com muito mais eficácia a interpretação do usuário:

- coração vermelho representando "vida do personagem";
- estrelas mostrando o número de policiais que estão perseguindo o personagem (que variam de zero até 6, no caso apresentado o valor é 2, através da cor amarela);
- semicírculo cinza identificando a intensidade que o colete a prova de balas agüenta;
- cifrão verde significando a quantidade de dinheiro do personagem.

Todas as informações transmitidas devem estar em harmonia, mantendo sempre um padrão associado ao tema do jogo e ao perfil do público alvo. O ambiente em que ele interage também deve apresentar esta harmonia. Ao mesmo tempo, deve mostrar o que o usuário nota, através da interface, que o sistema é capaz de realizar; impor os limites, até onde o usuário pode ir; e, o que este sabe ou precisa saber para progredir [Macedo Filho 2005]. É sempre necessário auxiliar no raciocínio lógico e intuitivo do jogador.

Além de todas essas preocupações, temos a presença das cores que são muito convenientes para auxiliarem na comunicação e reforçarem as emoções que se buscam passar ao jogador. Segundo Farina (1990), "as cores constituem estímulos psicológicos para a sensibilidade humana, influindo no indivíduo, para gostar ou não de algo, para negar ou afirmar, para se abster ou agir. As cores fazem parte da vida do homem porque são vibrações do cosmo que penetram no cérebro, para continuar vibrando e impressionando sua psique, para dar um som e um colorido ao pensamento e às coisas que o rodeiam; enfim, para dar sabor à vida, ao ambiente". Portanto, o cuidado com as cores é imprescindível na composição de uma interface. Harmonia e adequação aos temas são mais que necessários para que as cores também influenciem nas atitudes, interpretações e sensações do usuário. O fato das cores quentes, como vermelho e laranja, servirem preponderantemente para chamar a atenção, deve ser igualmente aproveitado dentro do jogo. Um exemplo de utilização das cores está presente na figura 5, através do vermelho utilizado no coração, cinza no semicírculo, amarelo nas estrelas e o verde no cifrão. Estas cores facilitaram a interpretação dos signos.

Como todo processo de comunicação, a troca de informações é indispensável para que este seja completo. Segundo Nielsen (1994), o sistema deve sempre manter os usuários informados sobre o que está acontecendo, através de um *feedback* adequado e no tempo certo. Dentro dos vários processos de comunicação que ocorrem num jogo, um deles é a resposta que o jogo dá ao usuário sobre determinada ação dele, ou pistas e dicas para que efetivamente aconteçam estas ações, isto é, o *feedback*. Um exemplo de *feedback* visual é apresentado na figura 5. No conteúdo grifado em vermelho, é passado ao usuário informações sobre o valor de bônus ganhos referentes às distâncias percorridas em determinado tempo.

Estas informações passadas do jogo para o usuário devem ter certos cuidados para que sejam bem interpretadas. Além de atender as leis da Gestalt, precisam ser expostas tempo suficiente para que o jogador as interprete e as assimile. Entretanto, esta exposição não deve ser muito longa, pois acaba atrapalhando ou levando o jogo à monotonia. É preciso que o desenvolvedor utilize uma linguagem simples e direta. O sistema deve falar a língua do usuário, com palavras, expressões e conceitos que lhe são familiares, em vez de utilizar termos orientados ao sistema. O projetista deve seguir as convenções do mundo real, fazendo com que a informação apareça em uma ordem natural e lógica, e seja interpretada rapidamente [Nielsen 1994].

A comunicação pessoal, com o passar do tempo, vem sendo cada vez menor, visto que a Internet e a televisão fazem com que as pessoas se relacionem diretamente menos. O mesmo acontece com os jogos. Para preencher esta necessidade de um contexto social são utilizados os personagens em jogos [Gulza e Haakeb 2005]. Eles são importantes para que haja uma socialização e identificação do usuário com o jogo, além de influenciarem no processo de aprendizado presente no mesmo. Com sua presença, é indispensável que tenha uma relação com o tema proposto pelo jogo, para que assim ocorra a formação de um processo de identidade do usuário com a interface e com o jogo em si. Ainda, segundo Gulza e Haakeb (2005), existem estudos que mostram que os usuários reagem diferentemente de acordo com a personalidade e a disposição dos traços físicos dos personagens. Logo, forma física, personalidade e movimento são questões a serem bem analisadas na criação dos personagens. É necessário, mesmo que minimamente, que o usuário possa selecionar algumas características ou os tipos físicos e de personalidade de seu personagem.

3.2.2 Parte Sonora

Conforme citado, a interface não se resume apenas à parte visual. Os jogos eletrônicos utilizam inúmeras formas de comunicação. Efeitos sonoros também são usados para comunicar ao jogador modificações no contexto, ou simplesmente para aumentar o teor emotivo das situações propostas [Hickmann 2006]. Porém, antes de mais nada, o usuário deve ter a liberdade de poder configurar suas preferências. A presença de uma possível configuração dos padrões sonoros, como volume, é indispensável. A busca pelo realismo nos jogos também é incessante. Para isso, o sincronismo das ações com seus determinados sons deve ser perfeito, assim como a qualidade e a clareza que são transmitidas essas mensagens sonoras, visto que são uma forma de comunicação e precisam ser interpretadas corretamente.

Os sons devem ser dispostos de uma forma na qual o usuário seja capaz de diferenciar o que é uma mensagem sonora indicando modificação no contexto e o que são apenas efeitos contribuintes para o realismo do jogo. Não pode haver uma sobrecarga e, consequentemente, uma poluição sonora. Cabe ao usuário fazer as associações necessárias para relacionar o som com seu significado. É a partir deste momento que ocorre um processo de aprendizagem do significado de cada efeito. Muitas vezes, estes sons devem vir acompanhados de uma mensagem visual para fortalecer esta relação. No entanto, o desenvolvedor não pode deixar margens para que as interpretações ocorram de maneira equivocada e venham a interferir ou ocasionar associações erradas de

regras ou ações no decorrer do jogo, conforme cita o autor [Hickmann 2006]. Freqüentemente, esta busca pelo realismo acaba apresentando certos efeitos sonoros com muita redundância, e é preciso ter cuidado, já que essa repetição tende a "irritar". Cada som deve ter um significado específico dentro de todo o contexto do jogo, mas este realismo deve ser observado para que não se torne poluição sonora ou sobrecarga de informações.

3.2.3 Parte Sensível

Todo o jogo deve se moldar à forma de atuação do usuário, para que todos os tipos de jogadores se sintam à vontade e, conseqüentemente, tenham maior nível de diversão. A parte sensível da interface é onde este enfoque deve ser redobrado. Quando se fala em sensibilidade na interface, refere-se à forma como as informações do usuário vão sendo transmitidas para a máquina, ou seja, os comandos e os controles do jogador. É a forma como o usuário emite as informações para o sistema, e através de qual acessório, teclado, mouse ou *joystick*, isto acontece. Para automatizar e facilitar as ações, também se faz importante a presença de atalhos específicos. Conforme inúmeras análises e observações, é necessário que o usuário seja capaz de configurar a sua maneira preferida de enviar estas mensagens (controlar o jogo), seja como isto vai ser feito ou por onde. Cada usuário tem uma preferência e um costume de como jogar, e cabe ao jogo ser capaz de moldar-se a estas variações de controle. O importante é o usuário sempre se sentir muito à vontade com a utilização do jogo, e conseguir da forma mais fácil possível interagir com mesmo.

3.3 Mecânica

Mecânica é o item responsável pela parte técnica do programa, também referenciado como a relação entre máquina e máquina, que avalia itens relacionados ao ambiente no qual o jogo se desenvolve, bem como suas respectivas regras. Em outras palavras, a mecânica se refere ao mecanismo do jogo, a forma com que ele lida com as interações do jogador.

[Fullerton et al. 2004] apresentam os seguintes elementos como os formadores da estrutura básica de um jogo, elementos estes que o levam a ser classificado como um jogo ao invés de um outro tipo de entretenimento:

- Jogadores
- Objetivos
- Procedimentos
- Regras
- Recursos
- Conflito
- Barreiras
- Desfecho

Em relação à mecânica de um jogo, é possível destacar quatro destes elementos:

- **Procedimentos:** métodos para jogar, que são constituídos de ações que os usuários podem realizar para concluir os objetivos do jogo;
- **Regras:** definem o universo do jogo, com as ações permitidas aos jogadores;
- **Recursos:** itens utilizados para se atingir o objetivo do jogo, valorizados pela sua escassez e utilidade;
- **Barreiras:** são os limites do jogo, podendo ser físicas ou conceituais.

Com foco nos elementos acima, e no intuito de organizar e agilizar o processo de avaliação de usabilidade da mecânica de um jogo, esta é dividida em duas partes: ambiental e condicional.

3.3.1 Mecânica Ambiental

A mecânica ambiental caracteriza-se por ser focada no ambiente virtual no qual o jogo se desenvolve. Nesta parte avaliam-se os recursos e as barreiras. Como exemplo de recursos, podem ser citados os itens que o jogador deve coletar em um jogo de estratégia (Figura 6), e que são vitais para seu sucesso no jogo.

Figura 6: *Age of Empires 2: The Conquerors* (Microsoft Game Studios/Ensemble Studios)

No caso das barreiras, é possível citar as quatro linhas que limitam um campo de futebol, que definem onde a ação do jogo ocorre.

3.3.2 Mecânica Condicional

A mecânica condicional caracteriza-se por ser focada nas ações do usuário. Nesta parte avaliam-se os procedimentos e as regras do jogo. Um exemplo de procedimento seria os passos que o jogador deve executar, dentro do menu, até iniciar o seu jogo. Já a possibilidade de um jogador poder agredir "virtualmente" outros jogadores, em um ambiente multi-jogador, é um exemplo de regra (Figura 7).

Figura 7: *Lineage II* (NCsoft)

Clanton [Clanton 1998] compara a avaliação de usabilidade da mecânica de um jogo à avaliação da funcionalidade de uma aplicação, por esta etapa ser mais dependente de um software pronto, quando a funcionalidade do jogo representa mais do que os conceitos do jogo. Mas, apesar de aproximar-se de um teste funcional, esta etapa da avaliação da usabilidade de um jogo deve ser focado na eficiência **de uso**.

4. Considerações Finais

Na pesquisa relatada neste trabalho, propõe-se verificar a usabilidade de um jogo eletrônico não genericamente, como é feito tradicionalmente com sistemas interativos, mas a partir da análise de suas especificidades, baseadas em sua jogabilidade, interface e mecânica.

Profissionais de diferentes áreas (psicologia, *design* e computação), baseados no que se encontra na literatura, desenvolveram *checklists* (listas de diretrizes) a serem utilizadas especificamente para avaliação de jogos eletrônicos. No geral, conta-se com um número de aproximadamente 150 diretrizes (entre as três *checklists*), as quais foram (e seguem sendo) trabalhadas e utilizadas para avaliar diferentes categorias de jogos eletrônicos, para fins de refinamento.

Além disto, foram organizados grupos de discussão com jogadores de diferentes perfis, a fim de verificar maiores contribuições para o método, a partir da experiência dos mesmos.

Outro objetivo desta pesquisa é verificar como estas diferentes *checklists* podem ser utilizadas (em separado e em conjunto) durante o processo de projeto e desenvolvimento de um jogo eletrônico. Ou seja, além de poder aplicar-se o conjunto das *checklists* a um jogo, quando o mesmo está prestes a ser lançado no mercado, *checklists* individuais (ou jogabilidade ou interface ou mecânica) podem ser aplicados desde o início de projeto, a fim de propiciar que problemas sejam detectados, e corrigidos, em tempo hábil (tempo de desenvolvimento).

A partir dos resultados já obtidos, acreditamos que a *checklist* de jogabilidade, por exemplo, possa ser aplicada no início do projeto do jogo, quando ainda estão sendo desenvolvidos o enredo e os desafios do mesmo, sem ter um protótipo funcional. E a mesma pode ser seguida pela de mecânica, quando as funcionalidades já começam a ser implementadas, e pela de interface, quando esboços de cenários e *layouts* são desenvolvidos.

Wixon (2006) diz que, dado o custo elevado de jogos, o crescimento deste mercado, seus ambientes de desenvolvimento flexíveis e os interessantes desafios que jogos apresentam, podemos esperar uma maior inovação em métodos e técnicas relacionadas à pesquisa na área de IHC. É neste campo, relativamente novo e em franca expansão, que esta proposta se insere.

Referências Bibliográficas

ALVES, L.R.G., 2005. *Game over: jogos eletrônicos e violência*. São Paulo: Futura.

ANDERSON, J.R., 2005. *Aprendizagem e memória: uma abordagem integrada*. 2. ed. Rio de Janeiro: LTC.

ARMSTRONG, T., 2001. *Inteligências múltiplas na sala de aula*. 2. ed. Porto Alegre: Artmed.

CLANTON, C., 1998. "An Interpreted Demonstration of Computer Game *Design*". In: *Proceedings of ACM International Conference on Human Factors in Computing Systems - CHI´1998*. pp.1-2.

COELHO NETTO, J.T., 1999. *Semiótica, informação e comunicação: diagrama da teoria do signo*. São Paulo: Perspectiva.

DIAS, C., 2003. *Usabilidade na Web: criando portais mais acessíveis*. Rio de Janeiro: Alta Books.

ERGOLIST, 2006. *Checklist para Avaliação Ergonômica* [online] UFSC/SENAI-SC/CTAI. Disponível em: http://www.labiutil.inf.ufsc.br/ergolist/index.html. [Acessado em 10/08/2006].

FARINA, M., 1990. *Psicodinâmica das cores em comunicação*. São Paulo: Edgard Blücher.

FEDEROFF, M. A., 2002. *Heuristics and Usability Guidelines for the Creation and Evaluation of Fun in Video Games*. Submitted to the faculty of the University Graduate School in partial fulfillment of the requirements for the degree Master of Science in the Department of Telecommunications of Indiana University. December 2002.

FULLERTON, T.; SWAIN, C. E HOFFMAN, S., 2004 *Game Design Workshop: Designing, Prototyping, and Playtesting Games*. Gilroy, CA: CMP Books.

GOMES FILHO, J., 2000. *Gestalt do Objeto: sistema de leitura visual da forma*. São Paulo: Escrituras.

GULZA, A.; HAAKEB, M., 2006. Design of animated pedagogical agents – a look at their look. *International Journal of Human-Computer Studies*, Vol. 64(4), pp. 281-394.

HICKMANN, F. O papel dos efeitos sonoros na significação em jogos eletrônicos. *Anais do ENCAM*. Curitiba: Editora da UFPR, 2006.

IwC, 2007. IwC, 2007. HCI Issues in Computer Games. *Interacting with Computers (IwC)*. Volume 19, Issue 2, Pages 135-304, March 2007.

JONES, G., 2004. *Brincando de matar monstros: por que as crianças precisam de fantasia, videogames e violência de faz-de-conta* (Tradução Ana Ban). São Paulo: Conrad Editora do Brasil.

KOSTER, R., 2005. *A theory of fun for game design*. Arizona: Paraglyph Press.

MACEDO FILHO, M.D., 2005. "A relevância de interfaces gráficas amigáveis para jogos eletrônicos funcionais". *I Seminário Jogos Eletrônicos, Educação e Comunicação: construindo novas trilhas*, UNEB, Salvador, out. 2005.

MACK, R. E NIELSEN, J. (EDS.), 1994. *Usability Inspection Methods*. New York, NY: John Wiley & Sons.

MEDLOCK, M. C., WIXON, D., TERRANO, M., ROMERO, R. L. E FULTON, B., 2002. *Using the RITE method to improve products: a definition and a case study*. Usability Professionals Association, Orlando FL, July 2002.

NIELSEN, J., 1993. *Usability Engineering*. San Diego: Academic Press.

NIELSEN, J., 1994. Heuristic Evaluation. In: MACK, R. E NIELSEN, J. (EDS.) *Usability Inspection Methods*. New York, NY: John Wiley & Sons, 25-62.

PRATES, R.O.; DE SOUZA, C.S. E BARBOSA, S.D.J., 2000. "A method for evaluating the communicability of user interfaces". *Interactions*. Vol.7, No.1, 31-38.

PREECE, J. (ed). *Human-Computer Interaction*. Harlow: Addison-Wesley, 1994.

RUBIN, J., 1994. *Handbook of Usability Testing*. New York, NY: John Wiley & Sons.

STERNBERG, R.J., 2000. *Psicologia Cognitiva*. Porto Alegre: Artmed.

WHARTON, C.; RIEMAN, J.; LEWIS, C. E POLSON, P., 1994. The Cognitive Walkthrough Method: A Practitioner's Guide. In: MACK, R. AND NIELSEN, J. (EDS.) *Usability Inspection Methods*. New York, NY: John Wiley & Sons, 105-140

WIXON, D., 2006. What work? *Interactions*, july + august 2006. 18-19.

Providing Expressive Gaze to Virtual Animated Characters in Interactive Applications

Rossana B. Queiroz
Leandro M. Barros
Universidade do Vale do Rio dos Sinos

Soraia R. Musse
Pontifícia Universidade Católica do Rio Grande do Sul

Figure 1: Snapshots of Alice's performance in our prototype.

Abstract

Eyes play an important role in communication among people. The eye motion expresses emotions and regulates the flow of conversation. Therefore, we consider fundamental that virtual humans or other characters, in applications such as Embodied Conversational Agents (ECAs), games and movies, present convincing aspects in their gaze. However, we perceive that in many applications which require automatic generation of the facial movements, such as ECA, the eye motion does not have meaning, mainly when gaze is related to character's expressiveness. This work proposes a model for automatic generation of expressive gaze, considering the eye behavior in different affective states. To collect data related to gaze and expressiveness, we made observations in Computer Graphics movies. These data were used as basis for expressions description in the proposed model. We also implemented a prototype and some tests were performed with users, in order to observe the impact of eye behavior in some emotional expressions. The results we obtained show that the model is capable to generate eye motion coherent with the affective states of a virtual character.

Keywords:: eye animation, expressiveness, gaze behavior, behavioral animation, facial animation

Author's Contact:

fellowsheep@gmail.com
leandromb@unisinos.br
soraia.musse@pucrs.br

1 Introduction

Nowadays, it is noticed that automatic gaze behavior generation in interactive application characters is still problematic in the sense of production of results that convincingly express their internal state and their communicative intentions. Several works [Cassell et al. 1997; Cohen et al. 2000; Fukayama et al. 2002; Lee et al. 2002] have researched ways to generate gaze behaviors coherent to conversational aspects and other ones [Poggi et al. 2000; Gu and Badler 2006] also explore cognitive and perceptive aspects. Few works, such as Poggi *et al.* [Poggi et al. 2000], try to correlate gaze behavior with agent's internal state aspects. Their focus generally is in the facial expressions (eyebrows, eyelids, wrinkles) and in the gaze direction during a given affective state, but they do not explore the manner how human eyes move differently according to their affective state.

On the other hand, we can observe in good animated Computer Graphics films that the characters's eye motion is extremely convincing, expressing in a clear way what the artists want to show. In this type of animation, every signal is intentionally and manually crafted by an animator to convey the desired meaning to the audience. It is not a trivial task[1], and animators adjust manually every gaze until they reach the desired result.

In this context, the proposed model in this work is mainly based on the hypothesis that Computer Graphics films can provide us a good parametrization of eye motion in function of character emotion. Obviously, it could be also observed in real life; however, we believe that computer graphics films are more useful in the sense that they use the same language we are dealing with. In order to do this, we follow a methodology inspired in Lance *et al.* [Lance et al. 2004] work, as it has a similar goal to ours. However, differently of Lance's methodology, our intention, for now, is not to generate accurate eye movements by precise eye measurements, but to identify behaviors and codify them into a computational system, in order to improve character's empathy. Consequently, interactive applications including convincing eye movements could be automatically generated, without artists intervention.

This work attempts to model different eye movements as gaze behaviors, that occurs acording to character's internal state as well in other communicative signals. We consider also some details that contribute with expressiveness as a whole and give more realism to the animation, such as the association of eyeballs, eyelids and head movements. This work explores the main elements that produce the expressiveness and interactivity in the human gaze and introduces this behavioral approach together with the statistical model of Lee *et al.* [Lee et al. 2002], in order to generate coherent eye motion automatically in interactive applications, such as ECAs and games.

The next section presents some related work. Section 3 provides complete description of model: the methodology we adopted to collect data (Subsection 3.1), the gaze behaviors we implemented (Subsection 3.3), how these behaviors were codified into the system (Subsection 3.2) and overall model architecture (Subsection 3.4). In Section 4 we present some obtained results and, concluding the paper, we draw some final remarks.

2 Related Work

Although the eyes play a fundamental role in non-verbal communication process among people, research on facial animation has not deeply focused on eye motion. There were several facial animation studies that relate affective states to expressiveness, but generally they do not include eye motion. We perceive, analyzing the bibliography, that the relation between eye motion and affective states was not much explored until now. Several work have proposed face-to-face conversation with users. Some of them explore

[1] For example, Making of Ice Age, Blue Sky (2002) shows the animators' efforts to produce extremely convincing gaze expressions in their characters, specially in the Mammoth Manfred, whose facial structure demands to show almost whole expression in its eyes.

default gaze behavior in a dialogue, such as mutual gaze and gaze aversion during talking and listening turns [Cassell et al. 1997; Cohen et al. 2000; Fukayama et al. 2002]. According to [Vertegaal et al. 2001], gaze may be used to discover who will assume the discourse in turn taking of conversation. Models such as [Cohen et al. 2000; Fukayama et al. 2002; Lee et al. 2002; Vinayagamoorthy et al. 2004] consider if the agent is in talking or listening state, in order to determine parameters that generate differentiated gaze. [Lee et al. 2002] work verified, through analysis of data collected by *eye tracking*, that these two states present different probability distributions in their eye motion intervals.

There are also approaches focused on the attentional focus of the agents, endowed with artificial vision abilities to provide gaze behavior when they are immersed in a virtual world [Itti et al. 2004] or in virtual reality applications [Courty et al. 2003]. The recent work of Gu and Badler [Gu and Badler 2006] tries to find patterns to develop a model that simulates ECA's gaze in a conversation environment, considering not only engagement but also distractions that can occur in a dynamic environment, where peripheral movements can affect the gaze behavior. This work also lists some gaze patterns for some emotional states and for cognitive and communicative tasks, based on literature.

Our work is deeply inspired in the following three works. The first is proposed by Lee et. al [Lee et al. 2002], that provides a stochastic model for generating saccades (rapid movements of both eyes from one gaze position to another). The four parameters for saccades generation are: *i)* magnitude angle of eyeball rotation, *ii)* direction (2D rotation axis), *iii)* duration (time of saccade execution) and *iv)* inter-saccadic interval (time of gaze fixation). These parameters are captured through an Eye Tracking system, which also considers if people are talking or listening. The results of this work are very realistic, but do not consider intentional communicative signals (related to text semantics) and affective states.

Our first efforts [Rodrigues et al. 2007] were to include this saccadic behavior together with facial expressions. It improves the realism in animation, but statistic model generates incoherent "gaze" in some cases, depending of the emotional expression. For example, some emotions require gaze fixation to the conversation partner, while others should produce gaze glances or gaze shifts that occur with more frequency. Indeed, saccadic behavior does not cosider emotion in this model.

The second work we were inspired in is Poggi et al. work [Poggi et al. 2000; Poggi and Pelachaud 2002], that focused on gaze behavior and proposed a meaning-to-face approach, allowing to simulate automatic generation of face expressions based on semantic data. Like any communicative signal, gaze includes two different aspects: a *signal*, which encompasses a set of physical features and the dynamic behavior of eyes in gaze; and the *meaning* which is related with a set of goals or beliefs that gaze intends to communicate. This work maps very well the different gaze functions and the relevant parameters that contribute to generate a convincing animation. In their proposed model, they classify the "meanings" by adding information about the world and agent's mind: *i)* agent's beliefs, *ii)* agent's goals and *iii)* agent's affective state.

Finally, Lance et al. work [Lance et al. 2004] addresses the problem of providing automatically generated expressive gaze to a virtual agent. Their approach is quite different of previous work, since they opt to capture gaze patterns data from animated CG films. They give two reasons for basing their work on films observations: *i)* literature does not provide sufficient data related to how eye shifts vary according to character's internal state; and *ii)* in animated CG films, gaze shifts are explicitly crafted to be understood by audience. Lance's work has good ideas, however it presents only preliminary data analysis and does not present a gaze generation model.

In our work, differently from the previous cited papers, the main goal is to provide a model to automatically generate expressive gaze with different gaze behaviors (gaze directions and shifts) and facial expressions (eyelids, eyebrows and head) based on empirical observation of CG films that explicitly evoke the meaning of agent's internal affective state. Our methodology of data collecting and its codification to a computational model are described in next section.

3 Model

We based our model mainly in three ideas:

1. the stochastic model of Lee [Lee et al. 2002], as an engine for generation of realistic gaze shifts;
2. the gaze *lexicon* from Poggi *et al.* work (gaze parameters) [Poggi et al. 2000], for mapping of signals and meanings of agent's internal state, and;
3. the idea of extracting gaze data from CG films, from Lance *et al.* [Lance et al. 2004] work.

This section presents the proposed model for automatic gaze generation. First, we talk about the methodology we adopted to collect gaze behavior data. Next, we describe the description language we used to translate collected parameters data in a computational system and the gaze behaviors implemented in our model. Finally, we present the overall architecture of the model and describe its modules.

3.1 Methodology

This section describes the methodology we adopted to observe gaze behaviors and collect gaze parameters data. The goal in the films observation is to identify gaze behaviors that reflect the character's internal state and their association with facial expressions.

The facial parameters we choose to be observed in film scenes (Table 1) are inspired on Poggi's list of gaze parameters presented in [Poggi et al. 2000], but we proposed some modifications in order to annotate gaze behavior variations:

- Observation of blink frequency: in a empirical way, we describe if blink frequency rate had some alteration with the observed affective state (slow, default or quick);
- Observation of saccadic parameters: also in high-level, we describe the range of shifts (magnitude) and inter-saccadic intervals (frequency) or information about a specific directed gaze (direction);
- For now, we are not considering data about brow "wrinkles";
- The field "extra features" can be used if we find some feature in behavior that can not be annoted in default parameters, for further studies;
- In our model, as we will further describe, head follows gaze shifts with great magnitudes, according to literature. But in some expressions, head movimentation can be intentionally differentiated. For this cases, we judge necessary to put also head's parameters as Table 1 shows.

In observed scenes, we must also identify the character's internal state, in order to map the facial expression and the gaze behavior. Identifying emotions and other affective states is not a trivial task [Ortony and Turner 1990; Ekman 1993; Cohn 2006]. Our guideline for emotion identification was the table of Emotion Types from Elliot [Elliott 1992] work. This work is strongly based on the OCC (Ortony, Clore and Collins) emotional model, which is the base for many agent's emotional models [Picard 1997]. In Elliot's work we found some definitions of emotions that helped us to identify the emotions in the observed scenes.

Our methodology followed these steps:

1. First, we choose films we consider to have convincing eye animation. We decided to observe three different animation languages, in order to "capture" a diversity of behaviors:
    - Cartoon-like humans (film choosen: The Incredibles[2])

[2] Pixar (2004)

Table 1: Observed parameters in selected scenes

Eyebrows: right/left eyebrow	
inner part	up / central / down
medial part	up / central / down
outer part	up / central / down
Eyelids: right/left eyelid	
upper	default / raised / lowered
lower	default / raised / lowered
Blink Behavior	
frequency	default / quick / slow
Eye Behavior	
humidity	dry / wet / tears
pupil dilatation	default / dilated / narrow
magnitude	default / great / small / none
direction	default / left / up-left / up / up-right right / down-right / down / down-left
frequency	default / quick / slow / fixation
some "extra" feature	free description
Head	
pitch	default / up / down
yaw	default / left / right
roll	default / left / right

- Realistic humans (film choosen: Final Fantasy: the Spirits Within[3])
- Cartoon-like non-humans (films choosen: Ice Age 1 and 2[4])

2. We watched the choosen films entirely, annotating the scene shots we considered of interest to our research (expressions and/or gaze behaviors), identifying the characters' emotional state, if pertinent. In this stage, for each film we created a table, containing the initial and final scene shot times, a brief scene description and the characters affective states identified in the scene.
3. We analysed annotations. From this analysis, we:
   - Identified (gave names to) the behaviors found. From these subjective analysis, we identified the four behaviors presented in Subsection 3.3;
   - Described and parametrized them (in a subjective way, describing "how it happens" and the facial parameters which contribute to animation);
   - Scanned in annotations the scenes that presented these behaviors, selecting them.
4. For each selected scene shot, we:
   - Described and parameterized according to our facial parameters (Table 1);
   - Translated this information to our description language (described in Subsection 3.2).

Finishing these steps, we must include our collected data (gaze behaviors and expressions) in a computational model which reproduces the gaze parameters described on an animated face.

3.2 Gaze Description Language

In order to codify collected gaze parameters into our facial animation system, we developed a simple language named *Gaze Description Language* (GDL). This language has two purposes: *i)* to describe facial parameters we extracted for the affective states in CG films, and *ii)* to provide the input for our gaze generator system with one or more gaze actions, in sequence. These sequences of actions could be supplied by an interactive application, such as a game or ECA system, or manually edited in a script file.

There are two types of GDL script files:

1. **Gaze Expression Description Files**: in our model, each emotional state captured from films shots annotations is codified into a description file, which contains collected information about eyebrows, eyelids, eyes and head. All of these description files, such as the example below, are part of the *Behavioral Database* module, which will be explained in the Subsection 3.4. The numerical parameters we see in example below describe intensity of parameter values. Values zero and one represent the inferior and superior thresholds of each parameter, which are defined into the *Expressive Gaze Generator* module of the model (Subsection 3.4).

```
Animation =
   {
      -- Eyebrows Info
      eyebrows = {
         inner = {
            { "l", "up", 1.0 },
            { "r", "up", 1.0 },
         },
         medial = {
            { "r", "down", 0.5 },
            { "l", "down", 0.5 },
         },
         outer = {
            { "l", "down", 0.3 },
            { "r", "down", 0.3 },
         },
      },

      -- Eyelids Info
      eyelids = {
         upper = { "lr", "lowered", 0.4 },
         lower = { "lr", "raised", 0.0 },
         blink = "quick",
      },

      -- Eyeballs Info
      eyeballs = {
         humidity = "wet",
         pupil = "default",

      -- Eye behavior
         behavior = {"distress", 0.9},
      },

      -- Head Info
      head = {
         pitch =  0.5,
         yaw = 0.0,
         roll = 0.0,
      },

      -- Mouth Info
      mouth = "none",
   }
```

2. **Gaze Storyboard Files**: are the action script files, providing a high-level interface for the automatic gaze generator system. In them, we describe one or more gaze parameterized actions. There are two types of actions we can define in these scripts: *i)* a pre-scripted affective state (facial expression) that exists in Behavioral Database and time (in frames) of animation to be generated; and *ii)* the eye behaviors implemented (without facial expression) in the model and time animation. If action is an eye behavior, facial expression (eyebrows, eyelids and head) is maintained neutral. The example below illustrates a simple gaze storyboard file. Note that the eye behaviors (described with details in Subsection 3.3) are the same whose are part of Gaze Expression Description Files. Each animation is automatically generated with the given time, according to expression and/or eye behavior annotated.

```
Storyboard = {
```

[3]Chris Lee Productions (2001)
[4]Blue Sky (2002 and 2006)

```
    {"default", "talking", 3000},
    {"sad", 1000},
    {"default", "listening", 1000},
    {"lookTo", "up", 17.0, 200},
    {"internalDialogue", 50},
}
```

3.3 Gaze Behaviors

Our proposed model implements different gaze behaviors. To generate "default gaze" (not related to character's affective state), we opted to use the Lee's model:

1. **Default Listening**: Lee's statistical model in listening mode;
2. **Default Talking**: Lee's statistical model in talking mode.

For directed specific gaze, we implemented an action we named *Look To*, whose parameters are the eye shift magnitude and direction and the total time of animation. Duration of the eye shift movement (saccade) is calculated by the statistical model formulae [Lee et al. 2002; Vinayagamoorthy et al. 2004].

From bibliography [Gu and Badler 2006; Brooker 2007], we implemented these six behaviors:

1. **Visual Construct**: upper-right gaze shift, with fixation. Behavior occurs when we try to envision an event that has never been seen;
2. **Visual Recall**: upper-left gaze shift, with fixation. Recalling an event that has been seen;
3. **Audio Construct**: right gaze shift, with fixation. When making a new sentence, for example;
4. **Audio Recall**: left gaze shift, with fixation. When remembering something what was said, for example;
5. **Kinesthetic**: down-right gaze shift, with fixation. Sorting out sensations of the body (related to feelings and emotional);
6. **Internal Dialogue**: down-left gaze shift, with fixation. Carrying on an internal conversation.

For these behaviors, shift duration is provided by Lee's model. The magnitude and total gaze duration time should be specified as parameters.

Based on films observations, we identified and implemented the following behaviors, that represent the main contribution of this work in the model:

1. **Concentration**: gaze shifts are smaller and less frequent. Affective states that requires a fixed gaze may be described with this behavior, such as angry, liking and surprise. This behavior also may be used to denote attention (mutual gaze) for a given time.

   First, this behavior calls the routines of the statistical model, in order to generate the saccade parameters. However, the magnitude and inter-saccadic interval are changed later, according to specifyed parametrization, in this way:

   - Magnitude: we limit the magnitudes in a range from 0° to 5°, specifying values in the range from zero to one. Zero value mantains the eyes in static way (0°) and the value one performs a normalization of the magnitude generated by the statistical model to 5°[5];
   - Inter-saccadic interval: a specifyed value in the range from zero to one increments from 0% to 100% the inter-saccadic interval generated by the statistical model.

2. **Discomfort**: gaze shifts are more frequent with great (glances, escapes) or small magnitudes (out of control sensation). Affective states which requires a loss of control sensation or confused/hidden feelings such as commotion, annoyance, nervousness, gratitude and fear should be described with this behavior.

   We parametrized this behavior as function of magnitude and inter-saccadic interval, in this way:

   - Magnitude: the range of magnitudes can suffer a decrement from 0% (mantains the normal range) to 99,95% of its value. To do this, we specify an intensity value from zero to one;
   - Inter-saccadic interval: the inter-saccadic interval may be decremented from 0% to 99,95% of its value, in the same way of the magnitude.

3. **Distress**: gaze direction tends to down. Affective states such as sadness and shame may be described with this behavior.

   We parametrized this behavior in function of gaze direction, increasing the probability of down-left, down and down-right directions be generated to 26.67% (80% of occurrences probability, while the other directions share likewise the remaining 20%), and using the statistical model to generate the other saccade parameters.

   We also implemented, to complement this behavior, an other behavior we named *Slide Down*, that generates a gaze sequence to down, down-right, down, down-left and down. This sequence creates an animation inspired in films observation, which can not be generated directly by statistical model. This "complementar" behavior can be actived on the *Distress* behavior, with 50% of occurence probability (verifyed on beginning of frame generation), if the given time in script is bigger than 300 frames (the time of these animation is fixed in 300 frames). This behavior may also be specifyed to occur in the gaze script file or storyboard file.

4. **Irony**: gaze direction tends to up. Internal states such as impatience and reproach may be described with this behavior.

   We parametrized this behavior in function of gaze direction, increasing the probability of upper-left, up and upper-right directions be generated to 26.67% and using the statistical model to generate the other saccade parameters. As we do in *Distress* behavior, we implemented a complementar behavior we named *Turn Eyes*. This behavior generates a gaze sequence to up-left, up, up-right and right, generating an animation observed in the films, which the characters "turn round the eyes". These animation has the fixed duration time of 40 frames.

For all these behaviors, we can specify the time of animation (shift and fixation) in frames on GDL script. We can use them without facial expression (directly on Storyboard file) or in specific expression GDL files.

3.3.1 Associated Movements of Head, Eyelids and Eyeballs

In order to increase the realism in animation, our model implements an association between head and eyes movements and also between eyes and eyelids movements. For head-eyes association, we used, as rule, an information we found in bibliography: horizontal gaze shifts greater than 25° and vertical gaze shifts greater than 10°produces combined head and eye movement [Gu and Badler 2006]. We opt to use this rule for the behaviors *Default*, *Look To*, *Visual Construct*, *Visual Recall*, *Audio Construct*, *Audio Recall*, *Kinesthetic* and *Internal Dialogue*. To the *Discomfort* behavior, we opt in do not move the head, in order to generate an out of control sensation to the eyes, we observed in films. The *Concentration* behavior generates saccades with, at most 5° of magnitude, thus these rule never will be applied. In the behaviors *Irony* e *Distress*, we observed that head moved more frequently, we opt to make the association independent of the magnitude value.

The association of the head parameters values (MPEG-4[6] facial animation parameters – FAPs [Ostermann 1998]) follows the relation

[5]The constant values defined to the parameters alterations were determined by empirical way, through the subjective films analysis

[6]Moving Picture Experts Group

presented in Equation 1,

$$fap\_value = magnitude \times m/27.5 \quad (1)$$

where m (an intensity multiplier) was fixed in 0.15 for the behaviors that follow the bibliography rule and *Irony*, and 0.5 for *Distress* behavior. Constant 27.5 is the maximum magnitude value generated by Lee's model.

In proposed model, eyelids follows the eyeballs in gaze directed to down, down-left, down-right, up, up-left and up-right. For up, up-left and up-right gaze, FAP value is determined according to Equation 2. Down, down-left and down-right gaze follows Equation 3 to determine the FAP value,

$$fap\_value = magnitude \times 0.5/27.5 \quad (2)$$

$$fap\_value = magnitude \times 0.7/27.5 \quad (3)$$

where 27.5 is the maximum magnitude value generated by Lee's model and the constant values 0.5 and 0.7 (intensity multipliers) are constant that were defided in empirical way.

For blink behavior, which are not the focus of this work until now, we implementeder three different frequencies, thar occur in fixed intervals: default (85 frames), slow (105 frames) and quick (25 frames).

3.4 Overall Architecture

The overall architecture of our system is represented in Figure 2. The model consists in the following modules:

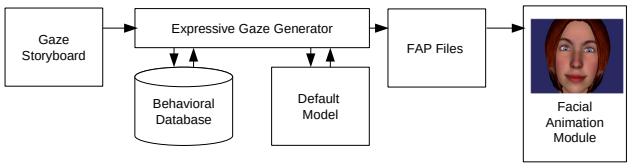


Figure 2: Overall architecture model diagram.

1. **Gaze Storyboard**: is the input of our system. This input can be of two types: *Gaze Storyboard* files (as descripted in Subsection 3.2) or direct calls to the routines of *Expressive Gaze Generator* module;

2. **Behavioral Database**: is the collection of gaze expressions, representing the affective states that were colected and descripted in GDL files. According to the *Gaze Storyboard* actions, this module provides the animation parameters for emotion expressions;

3. **Expressive Gaze Generator**: this module interprets the "command lines" (actions) of the *Gaze Storyboard* files, loads GDL files required from *Behavioral Database* and generates FAP (MPEG-4 Facial Animation Parameters) files. This module communicates with *Default Model* module to generate saccade parameters.

4. **Default Model:** is our implementation of Lee's statistical model, whose formulae are presented in [Lee et al. 2002] and and [Vinayagamoorthy et al. 2004] (to avoid negative numbers in velocities generation) and some other auxiliary methods that generate isolated parameters (respecting dependencies among themselves according [Lee et al. 2002]) when behavior requires alterations in magnitude, direction or inter-saccadic interval. Saccade duration always is calculated using Lee's model;

5. **Facial Animation Module:** is our visualization platform that follows MPEG-4 Facial Animation Standard [Ostermann 1998]. We built our Facial Animation Module using XFace [Balci 2004; Balci 2007], an open source toolkit for facial animation and research on talking heads.

   Next section presents our prototype system, that integrates all pieces of this model.

4 Results

This section presents the prototype we developed and an evaluation we made with users in order to verify the impact and acceptance of our gaze behaviors and then test the robustness of our model.

4.1 Prototype

The proposed model was implemented in a prototype system we called *Alice's Eyes Adventures in OSGLand*[7] The prototype was developed in Linux, using C++ language [Stroustrup 2000]. The Lua language [Ierusalimschy 2003] was also used for the GDL script files. For the visualization module, we opt to use the facial animation libraries of the Xface toolkit[Balci 2007], that is developed in C++ language and uses OpenGL API [Woo et al. 1999] for rendering.

We opt to build the system interface using the Xface Core (only the facial MPEG-4 animation libraries) on the Open Scene Graph (OSG) [Osfield 2007] environment. From these integration resulted a library (beta version) we called OSGXface. Thus, *Facial Animation Module* of our model is built on these library. The prototype, that includes the implementation of all model modules has the appearence illustred in Figure 3. In its interface, we can combine facial expressions of some affective states (*angry*, *disgust*, *fear*, *happy*, *sad* and *surprise*) with the behaviors *Default Talking*, *Default Listening*, *Concentration*, *Discomfort*, *Distress* and *Irony*, generating animations in real-time with the choosen emotion and gaze behavior or to open other GDL storyboard files to generate animations with other expressions and behaviors supported by the model, as showed in Figure 1.

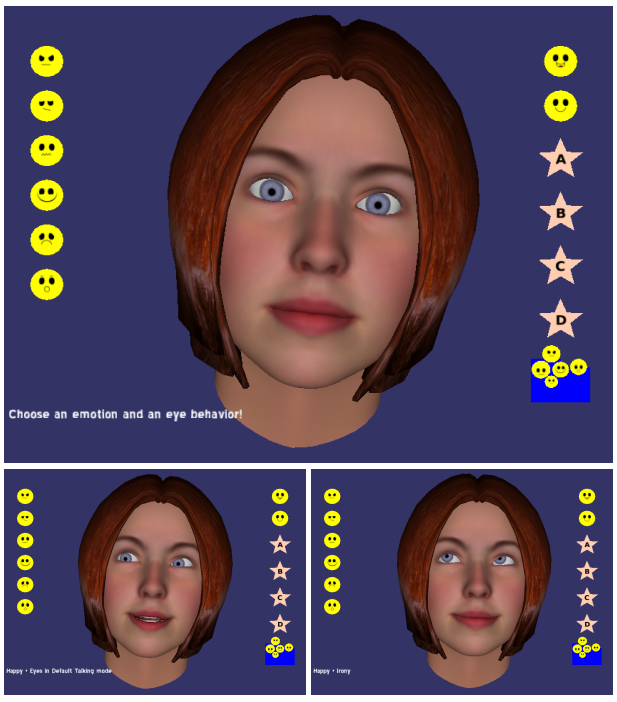


Figure 3: Screenshots of the developed prototype.

Altough this prototype interface is very simple, developed only with the purpose of to show the model capabilities and evaluate them, it

[7] Alice is the virtual human we used (part of Xface) which was introduced, in this prototype, in Open Scene Graph (OSG) environment.

can be considered also a framework to the development of applications that needs to include interactive faces, such as some games and ECA systems.

4.2 Gaze Behaviors on Prototype

The gaze behaviors presented in Subsection 3.3 were implemented on prototype and several storyboard files and expression description files was edited (fulfilling the *Behavioral Database*), in order to generate animations with these behaviors. Figures 4, 5 and 6 show some animation snapshots generated with the gaze behaviors *Default*, *Look To* (without facial expression annotations) and with facial expressions and gaze behaviors annotated in GDL scripts.

Following the methodology presented in Section 3.1, four film scene shots were selected. From these four scene shots (each one representing one of the four behaviors proposed in this work) we created videos with the animations generated by the prototype, in order to evaluate the impact of the implemented behaviors in users, described in the next subsection. Table 2 and Figure 7 identify the selected four scene shots we choose for the tests and show some snapshots of the generated animations.

4.3 Evaluation with Users

In order to evaluate the results obtained by the model, we opted in to do a subjective evaluation with users. This evaluation was realized through a questionnaire, via web, with the purpose of obtain an acceptable number of participants and collect opinions from an assorted audience.

For these evaluation, 12 videos became available to be downloaded by the volunteers before to answering the questionnaire, as described below (users did not know what method of eye animation was used in each animation):

Set *A*
- **a.avi**: the inspiring scene (scene shot) to the animations a1 and a2
- **a1.avi**: animation generated with the *Default* gaze behavior
- **a2.avi**: animation generated with the *Concentration* gaze behavior

Set *B*
- **b.avi**: the inspiring scene to the animations b1 and b2
- **b1.avi**: animation generated with the *Discomfort* gaze behavior
- **b2.avi**: animation generated with the *Default* gaze behavior

Set *C*
- **c.avi**: the inspiring scene to the animations c1 and c2
- **c1.avi**: animation generated with the *Distress* gaze behavior
- **c2.avi**: animation generated with the *Default* gaze behavior

Set *D*
- **d.avi**: the inspiring scene to the animations d1 and d2
- **d1.avi**: animation generated with the *Default* gaze behavior
- **d2.avi**: animation generated with the *Irony* gaze behavior

The questionnaire was divided in four parts, with different goals to evaluate:

Part 1 Evaluation of the generated animations. All generated animations (with *Default* and the four proposed behaviors) was evaluated by the users, that gave to each of them one of these ratings:

1 (Very Weak) | **2** (Weak) | **3** (Regular) | **4** (Good) | **5** (Very Good)

In this stage, the users answered 8 questions, to evaluate individually each one of the 8 generated animations:

Q1 – Q8: Give a concept from 1 to 5 for the Alice's "performance" in video (a1, a2, b1, b2, c1, c2, d1 and d2).

Part 2 Evaluation of affective states identification and relation to gaze behaviors. In this stage, users answered the following 4 questions, indicating what animation Alice seemed to present some specific affective state:

Q1: Was there some animation in which Alice seemed to be more concentrated, looking fix to you? In which?

Q2: Was there some animation in which Alice seemed to be more apprehensive, uncomfortably? In which?

Q3: Was there some animation in which Alice seemed do not be very well, sad or shamed? In which?

Q4: Was there some animation in which Alice seemed to be impatient, or talking ironically? In which?

Part 3 A set of more generic questions, in order to evaluate the impressions of "life", attention and emotion expressiveness of the character in the user:

Q1: Did Alice seem to have some "life" (5) or did you consider she extremely apathetic (1)?

Q2: Did she seem to have some interest in "talk to/listening you" (5)? Or did she seem do not be in a conversation (1)?

Q3: Did she seem to express some emotion (5) or did she seem do not have feelings (1) while talk or listen you?

Part 4 Free writing questions, in order to evaluate model and animation as a whole:

Q1: Did some animation seem incoherent to you, in the sense of the expression which Alice seemed to be feeling? (In other words, had some animations that you really did not like? Which? Why?

Q2: Was there some animation that called more your attention, in the sense that she seemed to be demonstrating in a clear way her feelings? (In other words, had some animations that you really liked? Which? Why?

Q3: Alice is a prototype of a talking head. If you should improve something in her face motion (head, eyes, eyebrows, mouth, nose, brow, cheeks), what did you do first?

Twenty one volunteers answered the questionnaire. Results of the Part 1 are expressed in Figure 8, that shows the percentage of the ratings attributed by the users to each one of the generated videos and Figure 9 shows a comparative between the means obtained by *Default* behavior and the four behaviors proposed in this work.

Analyzing the graphics, we perceive that all animation sets (A, B, C and D) obtained a mean score quite close. Generally speaking, the ratings attributed were quite positive ("Regular" to "Very Good"), except in the set D, which received a significant percentage of ratings "Weak". We believe that one of the reasons of these is explained by more than one user in the free writing questions (Part 4): the inspiring animation D is too short (2 seconds, approximately) and quite dark (it is night and raining in the scene). For this reason, some people related to have difficulties in understand what was happen in the animation d.

We also perceive in the graphics of figures 8 and 9 that the animation c1, which was generated using the *Distress* behavior was evaluated as the more convincing of the animation sets. It can be also confirmed in free writing questions, where more than one volunteers considered it the "most expressive" (question 2 of Part 4).

The volunteers demonstrate quite controversial in relation to the animations b1 and d2 (*Discomfort* e *Irony* behaviors). Note that, in the graphic of their score and in comparative of the score means,

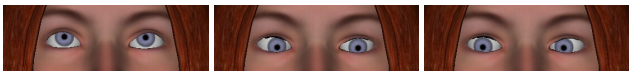


Figure 4: Animation snapshots running with *Default* behaviors.

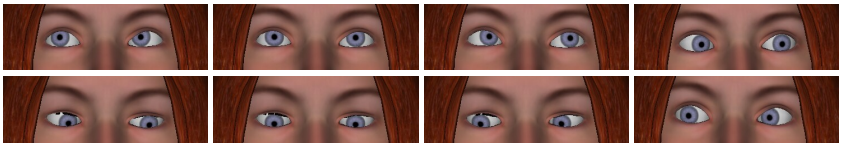
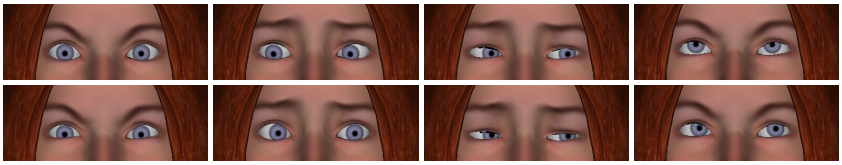


Figure 5: Animation snapshots running with *Look To* behavior, without facial expression.

Figure 6: Animation snapshots running with GDL scripts (including expressions and gaze behaviors).



a) b) c) d)

Figure 7: Animations snapshots with the gaze behaviors: a) *Concentration* b) *Discomfort* c) *Distress* and d) *Irony*

Table 2: Technical information about data we based to generate animations illustrated in Figure 7.

Label	Scene from film	Start time	End time	Affective state	Eye behavior
a)	The Incredibles	00:29:23	00:29:25	Angry	Concentration
b)	The Incredibles	01:00:49	01:00:52	Fears-confirmed	Discomfort
c)	Ice Age 2	01:17:27	01:27:34	Shame	Distress
d)	Ice Age	00:13:05	00:13:06	Arrogance	Irony

these animations obtained ratings lower than the animations generated with the *Default* behavior (b2 and d1). Analyzing the graphics together with the free writing questions, we perceive that some volunteers found the eyes behavior "strange" in these two animations (rapid frequency in b1 and the "turn round" of the eyes in d2). However, we can observe in the Part 2 of the questionnaire evaluation (commented below) that when the volunteers need to relate one animation to the asked affective states, these two "controversial" animations were the most indicated for the affective states that they really should represent.

Figure 10 shows the results obtained in the 4 questions of the Part 2 of questionnaire, which consists in the evaluation of the identification of some affective states by the users in the observed videos. Analyzing these graphics, we perceive that animations a2, b1, c1 and d2 (generated with the gaze behaviors *Concentration*, *Discomfort*, *Distress* and *Irony*, respectively), were the most indicated as representative of the asked affective states. These positive results indicate that there were coherence not so in facial expressions, but that the gaze behavior contributed in the identification of the affective states.

Figure 11 presents the results obtained in the 3 questions of the Part 3 of questionnaire, that evaluates some aspects more general of the animations. The evaluated aspects were: "how much" does Alice seems to have "life" in animations (Q1), "how much" does she seems to be attentive on conversation (Q2) and "how much" she seems to express her feelings (Q3). Note that, for all these items, the ratings attributed by the users were also positive ("Regular" to "Very Good", in at least 90% of the evaluations in each question. Regarding expressiveness, the results show that Alice seems convincing in her emotions expression to the volunteers (95.24% of the answers were "Good" and "Very Good", while the remaining was "Regular").

The free writing questions (Part 4) were elaborated in order to the participants could freely write their opinion. The given answers show aspects very important that suggests improvements as in Alice's 3D model as in animations. The items below present some of these considerations, that are more related to the scope of this work and deserve distinction:

- Concerning the eyes, the users suggest to implement the pupil dilatation and contraction;
- Head motion sometimes are too much sudden; users suggest improvements;
- Users suggest also the exploration of wrinkles on the brow, in order to accentuate the expressiveness;
- The eyebrows motion could be improved, since they remains static after the setting of expression movement.

We also present below some other users' suggestions and critics. Although they are not directly linked to the scope of this work, they are relevant to the development of a robust facial animation system or a more convincing talking head:

- Users suggest improvements in mouth motion (including the teeth, that do not follow the lips) and cheeks;
- One volunteer observed that, in his opinion, the hair covering the ears suggests that "Alice does not want to listen the user".

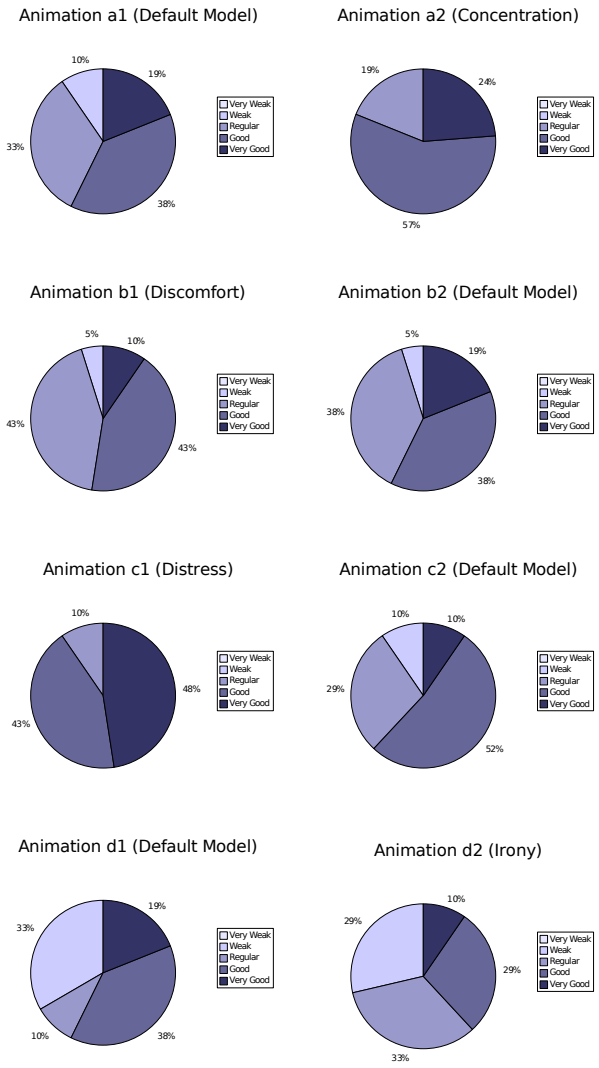


Figure 8: Results of subjective evaluation of the Part 1 of questionnaire (questions Q1 to Q8): the graphic shows the percentage of the ratings attributed by the users to the 8 generated animations.

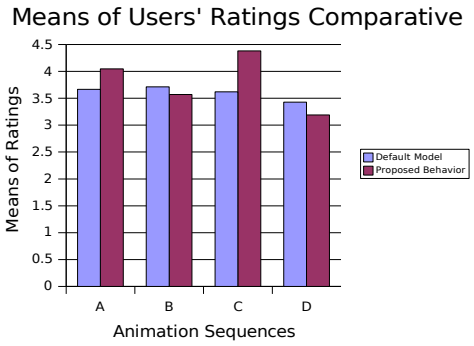


Figure 9: Results of subjective evaluation of the Part 1 of questionnaire: the graphic shows the mean score to each question, comparing the *Default* gaze behavior (statistical model) with the four proposed behaviors.

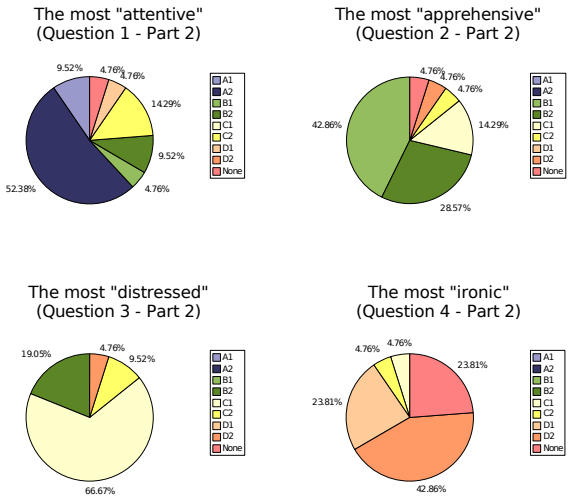


Figure 10: Results of subjective evaluation of the Part 2 of questionnaire: the graphic shows the percentage of the answers given by the users to each question.

5 Final Remarks

Our work presents a model for automatic generation of expressive gaze, applied to a MPEG-4 Facial Animation standard compliant visualization platform. To develop this model several studies were conducted, and data was collected in order to establish relations between affective states, expressiveness and gaze behavior.

Our first efforts [Rodrigues et al. 2007] resulted in the inclusion of Lee's saccadic behavior together with facial expressions, and an evaluation of the impact of these results. Although this technique can improve realism in animations, the statistical model can generate, due to its own nature, incoherent "gaze" patterns, related to the emotional expressions. For instance, some emotions require gaze fixation to the interlocutor, while others expect quick glances or gaze shifts. Indeed, the probability distributions obtained through Lee's experiments didn't consider emotional states, generating only a "default" behavior for listening and talking modes. As such, this model proposed some differentiated gaze behaviors, to be applied together with facial expressions, in order to bring more coherence to the animations.

The main results of our work are the data collection methodology, and our model that includes different gaze behaviors. These results constitute an automatic eye animation framework, which can be used in several interactive applications, such as ECAs and games. Note that the proposed eye animation model *per se* is not dependent of the MPEG-4 Facial Animation standard, and thus it can be used in different platforms. Moreover, the data collected to integrate the *Behavioral Database* module (described in high-level via GDL – *Gaze Description Language*) can be used in other applications requiring expressive faces (also independent of the animation platform). We can also underline, as technical contributions, the GDL prototype (which can be easily extended), some improvements made on Xface, as well its integration with OSG environment in our viewer prototype, which allows to work with animated human faces in the OSG environment.

Through the inclusion of gaze behaviors inspired in CG films observations, we believe we present a differential in relation to the other eye animation models found in bibliography. Moreover, by analyzing the users' opinions in our evaluation procedure, we understand that these behaviors need to be applied in the "right context" (in other words, they need to be integrated to facial expressions that correspond to the character's affective state). For example, we noted, via the scores given to the animations in group D (presenting arrogance, in an irony tone), alongside some critic in the free writing questions, that the inspiring animation (film scene shot) was too short and dark. This might become difficult to comprehend the character's affective state. For this reason, some people found that the way Alice's eyes shifted in animation d2 was "a bit odd". Others criticized b1 (animation generated with the *Discomfort* behavior), once Alice's eyes seemed to be "out of control". Indeed, the *Irony* and *Discomfort* behaviors (where the eyes behave very differently from the "standard" behavior) received the lowest evaluation

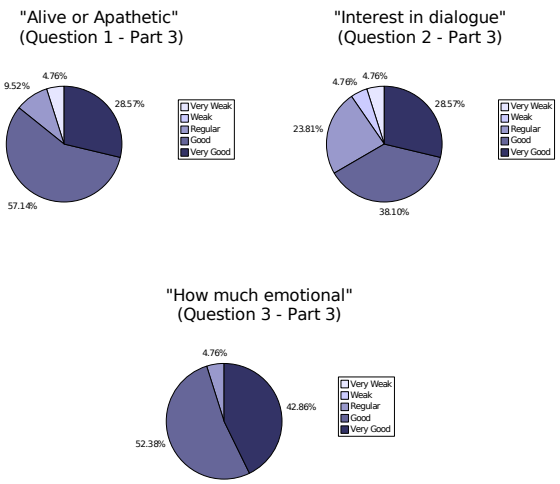


Figure 11: Results of subjective evaluation of the Part 3 of questionnaire: the graphic shows the percentage of the ratings given by the users to each question.

scores. However, in the second part of the evaluation, where the volunteers were asked in what animation Alice seemed to present a specific affective state, these two animations (b1 and d2) were cited as the most faithful to the affective states they should be representing. Thus, we conclude that we must be careful in experiments of these type in order to ensure that the volunteers can understand the animation context (what is being evaluated), so that no misunderstanding occurs.

Generally speaking, we noted that, in order to conduct this kind of evaluation, the quality of the visualization interface is very important: the users approved Alice's visuals, and made some constructive criticism, as shown in Section 4. We noticed that the problematic of eye movement (in the perception of the general public) is subtler than the facial expressions. However, this fact does not diminish the relevance of this study in any way. Results related to Part 2 of the questionnaire show that volunteers usually indicated the animations showing differentiated eye behavior as being the most expressive. This confirms the contribution of eye movement to general expressiveness.

When applying our methodology, we also identified some problems that could be explored as future work, in order to improve the proposed model:

- The identification of the character's affective states in the film scene shots is a complex task, since in dynamic scenes, both mixed emotions and task-related gaze commonly occur. Moreover, we did not observe in this work the influence of characters' *individual* features, such as personality, age, gender and culture;
- Measuring and quantifying the parameters in film sequences is not a simple task. We opted, in this work, to describe the parameters in a high-level way. However, the parameters *intensity* in our model continued to be defined in a empirical way, as in the presented related work [Poggi et al. 2000] (except parameters generated by the Default Model). We thought, as an alternative, to explore the use of Image Processing techniques in order to perform some gaze measurements, but early we discarded it, since all characters are extremely dynamic and their body morphology is assorted (different shapes and proportions of the head and eyes, for example). In this context, we believe that using motion capture techniques to collect accurate gaze parameters can be more easily done with humans;
- The observed films were chosen with the intention to cover three different kinds of character representations: realistic humans, cartoon-like humans and cartoon-like non-humans. We observed that the cartoon-like characters were more adequate for our purposes, because in the cartoon language, the stylized movements provided an easier gaze behavior identification. In the film with realistic humans that we observed (Final Fantasy), we supposed that there were an excess of care to maintain face features faithful with real humans, and so facial expressions seemed quite artificial (few muscle alterations). Moreover, since characters' eyes are small (human proportion), the observation becomes very difficult in scenes where the focus is not the character's face. We needed to be careful with Alice's face calibration parameters in each proposed gaze behavior, since she is also very realistic. For this reason, we suggest tests with other faces and studies to improve and to extend the model, in order to verify if FAP calibration needs alterations according to character's representation language.

Acknowledgements

Thanks to Koray Balci (Xface developer), that contributed so much in prototype development stage.

References

BALCI, K. 2004. Xface: Mpeg-4 based open source toolkit for 3d facial animation. In *AVI '04: Proceedings of the working conference on Advanced visual interfaces*, ACM Press, New York, NY, USA, 399–402.

BALCI, K., 2007. Xface. Available in `http://xface.itc.it/`. Last access: June 9, 2007.

BROOKER, D., 2007. Body language – eyes. Available in `http://www.hash.com/users/threechikens/eyel.html`. Last access: April 28, 2007.

CASSELL, J., TORRES, O., AND PREVOST, S. 1997. Modeling Gaze Behavior as a Function of Discourse Structure. In *First International Workshop on Human-Computer Conversations*.

COHEN, M. F., COLBURN, R. A., AND DRUCKER, S. M. 2000. The role of eye gaze in avatar mediated conversational interfaces. In *Technical Report MSR-TR-2000-81*, Microsoft Corporation.

COHN, J. F. 2006. Foundations of human computing: facial expression and emotion. In *ICMI '06: Proceedings of the 8th international conference on Multimodal interfaces*, ACM Press, New York, NY, USA, 233–238.

COURTY, N., BRETON, G., AND PELÉ, D. 2003. Embodied in a look: Bridging the gap between humans and avatars. In *IVA*, 111–118.

EKMAN, P. 1993. Facial expression and emotion. *American Psychologist 46*, 9 (April), 384–392.

ELLIOTT, C. D. 1992. *The affective reasoner: a process model of emotions in a multi-agent system*. PhD thesis, Evanston, IL, USA.

FUKAYAMA, A., OHNO, T., MUKAWA, N., SAWAKI, M., AND HAGITA, N. 2002. Messages embedded in gaze of interface agents — impression management with agent's gaze. In *CHI '02: Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM Press, New York, NY, USA, 41–48.

GU, E., AND BADLER, N. I. 2006. Visual attention and eye gaze during multiparty conversations with distractions. In *IVA*, 193–204.

IERUSALIMSCHY, R. 2003. *Programming in Lua*. Lua.org, Rio de Janeiro, Brazil.

ITTI, L., DHAVALE, N., AND PIGHIN, F. 2004. Realistic avatar eye and head animation using a neurobiological model of visual attention. *SPIE Applications and Science of Neural Networks, Fuzzy Systems, and Evolutionary Computation VI 5200*, 64–78.

LANCE, B., STACY, M., AND KOIZUMI, D. 2004. Towards expressive gaze manner in embodied virtual agents. In *Autonomous Agents and Multi-Agent Systems Workshop on Empathic Agents*.

LEE, S. P., BADLER, J. B., AND BADLER, N. I. 2002. Eyes alive. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 637–644.

ORTONY, A., AND TURNER, T. 1990. What's basic about basic emotions? *Psychological Review 97*, 315–331.

OSFIELD, R., 2007. Open scene graph. Available in `http://www.openscenegraph.org/`. Last access: June 9, 2007.

OSTERMANN, J. 1998. Animation of synthetic faces in mpeg-4. In *CA '98: Proceedings of the Computer Animation*, IEEE Computer Society, Washington, DC, USA, 49.

PICARD, R. W. 1997. *Affective computing*. MIT Press.

POGGI, I., AND PELACHAUD, C. 2002. Signals and meanings of gaze in animated faces. In *Language, Vision and Music*, Mc Kevitt, Paul, Seán Ó Nualláin and Conn Mulvihill, 133–144.

POGGI, I., PELACHAUD, C., AND DE ROSIS, F. 2000. Eye communication in a conversational 3d synthetic agent. *AI Communications 13*, 3, 169–182.

RODRIGUES, P. S., QUEIROZ, R. B., BARROS, L., FEIJÓ, B., VELHO, L., AND MUSSE, S. R. 2007. Automatically generating eye motion in virtual agents. In *IX Symposium on Virtual and Augmented Reality (SVR2007)*, 84–91.

STROUSTRUP, B. 2000. *The C++ Programming Language*, special third ed. Addison-Wesley, Reading, USA.

VERTEGAAL, R., SLAGTER, R., VAN DER VEER, G., AND NIJHOLT, A. 2001. Eye gaze patterns in conversations: there is more to conversational agents than meets the eyes. In *CHI '01: Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM Press, New York, NY, USA, 301–308.

VINAYAGAMOORTHY, V., GARAU, M., STEED, A., AND SLATER, M. 2004. An eye gaze model for dyadic interaction in an immersive virtual environment: Practice and experience. In *Computer Graphics Forum*, vol. 23.

WOO, M., DAVIS, AND SHERIDAN, M. B. 1999. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 1.2*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.