A Fast and Safe Framework to Prototyping Physical Worlds Using XNA and GPU

Jose Ricardo da S. Junior
Media Lab - UFF

Esteban W. Clua
Media Lab - UFF

Paulo A. Pagliosa
DCT - UFMS

Anselmo Montenegro
Media Lab - UFF

Leonardo Murta
Media Lab - UFF

Abstract

Computer games are becoming an important resource for many educational purposes. The increase of computational power and new paradigms of engine architectures not only reinforce this, but also make available new approaches to real time applications. It is also well known that interactive environments are strong mechanisms for a more efficient content comprehension and assimilation. One of these fields is physics, where experimentations and real time interactions are desirable. Although there are many software and computational environments focused on physics learning, few programming tools offer the facility for programming applications and experiments. This work presents a novel framework for physics programming that combines the facilities available in Microsoft XNA framework with C# and GPU acceleration. The present framework implements a complete set of functionalities for dynamic simulation of rigid bodies, encapsulating them completely with XNA.

Keywords:: Education for physics, XNA, CUDA, GPU aceleration

Author's Contact:

{jricardo,esteban,anselmo,leomurta}@ic.uff.br
pagliosa@dct.ufms.br

1 Introduction

A considerable amount of the power increase in computer technologies is pushed by the video-games and interactive entertainment industry, which aims at more and more realistic virtual experiences for the players. Due this, games and rich environment simulations are applications that demand complex tasks to be developed. In fact, they are composed of various computationally intensive problems, such as artificial intelligence, physics simulation, scene database query, networking, audio, video, I/O, etc.

Beyond entertainment, it is well known that video-games can have many applications, such as education [Papastergiou 2009], training and health. An example of this is the American's Army game, created by the United States Army with the purpose of providing an experience in the US Army tasks, focused for civilians' usage [Strong 2002].

Computer games and simulation is proved to be extremely efficiently at all stages of education not only for reinforcing knowledge but also for archieving high-level cognitive, affective and psychomotor objectives [Papastergiou 2009]. With this in mind, Microsoft Robotics [Microsoft 2008], a commercial platform for developing robotic applications, was developed, targeting students and researches on robotics, given them a framework that is capable of prototype a simulation easily in a few period of time [Workman and Elzer 2009].

The increase in computational power is making it possible to apply more efficient real time physics calculations. Simulations that were impossible to be executed in real time are now becoming possible [Yeh et al. 2006] and a large number of libraries and frameworks for this purpose are becoming available. Beyond the direct application on game development, these tools are also becoming an appropriate mechanism for physics teaching and research by educators. At this moment, most of these libraries can simulate rigid bodies, soft bodies and fluids, which act according to parameters that are set by the user in the simulation. Unfortunately, the usage of these libraries is not a trivial task and sometimes requires a lot of effort.

While some frameworks are purely code and allow programming almost any situation, some of these physics simulation frameworks are implemented using a visual interface, allowing an interactive feed-back of the application [Intel 2000]. The first situation requires highly specialized programming skill and the second brings many restrictions for the experiments and educational process. An alternative can be the direct usage of a commercial game [Price 2008] to build an educational experiment, but the constraints are stronger than using the visual interface of the engine itself, as most commercial games were not mainly designed to focus on education. In this case, making a simple physics simulation could be hard or even impossible as most of the physical related attributes are hidden from the user due they irrelevance in the game.

Another option can be the use of a commercial game engine that hides most of the low level programming from the user like is the case of [Unity 2004]. In this case, experiments could be made easily from researches at cost of numerical precision, as most game engines needs to deal with many subsystems like sound, artificial intelligence and networking which are hardly ever necessary for experiments research.

For complex physics simulation, integration with some API, such as OpenGL, DirectX, and CUDA, is required in order to achieve satisfactory and interactive results. However, this approach would increase even more the complexity for the educators, due to tasks that require expertise in systems architecture and computer programming. In this case, an educator that has a good knowledge in physics would also need skills related to computer graphics and GPU programming, loosing focus on the main educational purpose.

Many of these physics libraries are based on C/C++, since they require a lot of optimizations and are very computationally intensive. These classes of languages are called unmanaged languages, and are much harder for a simple user. They present complicated housekeeping details and development difficulties that are not related to the intended application, which are time consuming and error prone tasks. Another issue related to this kind of languages is its compilation time that can be too long depending of the application's complexity and size. For these reasons, the use of script languages like Lua [Puc-Rio 1993] is an obligation to avoid application recompilation and increase productivity.

In this paper we present a framework that gives educational software developers and educators with minima experience with programming an appropriate environment that facilitates the production of educational content, in particular on physics. The tools developed by this framework can also be used by research groups. This solutions intends to be a fast, easy, and safe way for exploring physics using computer graphics and interactivity. As it is an extension of the Microsoft XNA [Microsoft 2004] framework, it inherits its facilities. The proposed framework was developed with the following requirements:

- Memory safety, avoiding the user to worry about issues that are not related to the application itself;
- Fast interactivity and speed for showing the results to the user when executing the application;
- A fast physics end very efficient simulation that runs on GPU instead of CPU;
- Easy to use, even for non-programmers;
- Expansibility, allowing the development of other kinds of applications in the future;
- Modularity;

By adopting XNA as the base for the application, an efficient memory safety, ease and minimal compilation time is achieved, since these are benefits allowed by the C# language. We present a discussion about the efficiency related to unmanaged languages, like C++.

Choosing the Microsoft XNA and C# language was important for handling different kinds of input devices, graphics, sound, and network tasks easily. Due to its rich documentation and good software architecture, our proposed tool may be easily extended for more complex environments related to different specific physical educational processes. We built the framework completely based on Nvidia CUDA [NVidia 2004]. CUDA is a computer architecture that uses a C like programming language. Using this, the GPU can be fully explored as a parallel processor, allowing a very fast and efficient approach.

The rest of the paper is organized as follows. Section 2 discusses related work, section 3 presents the framework architecture and its core components and section 4 explains the usage of these components. Section 5 presents the results. Finally, section 6 concludes the paper.

2 Related Work

Many efforts had been made in order to facilitate the development of real time physics simulation by end users. One of the main objectives consists on hiding low level aspects to the developer, like file manipulation, direct low level programming, and graphics integration, among others. Following these criteria, Kačić-Alesić [Kačić-Alesić et al. 2003] proposed a plug-in that could do rigid and deformable body dynamics, particle dynamics, and hair and cloth simulation at a basic level. Although this work presents an elegant architecture, it still needs some kind of integration with the application, assuming that the user has a minimum knowledge of low level computer programming.

Shapiro [Shapiro et al. 2007] developed a toolkit that enables the users to create dynamic controllers for articulated characters under physics simulation, mixing pose animations with physically based motion. To prevent the user from using low level languages, this work provides script languages for interaction with the simulation.

In [Popović et al. 2000], the authors propose a simulator where the user can give the initial position, velocity, and final position of the simulated object. The simulator automatically generates physics based motion based on the input constraints. This simulator is easy to be used by end users but unfortunately is not capable to make different physics simulations.

PhysX [NVidia 2008] is a powerful physics engine that can be used with GPU acceleration, but requires knowledge of low level computer programming and requires graphics integration knowledge from the user for graphical feed-back. In some cases, a script language is also needed to avoid the simulation recompilation in case of parameters changing in the application.

VPython [Scherer 2000] is a framework that uses the Python programming language for creating 3D simulations and is used by researches in various scientific fields including Physics research. Although easy to use and fast interactively to show the simulation results, a simulation that requires a high physics processing and/or deal with a large amount of data maybe not be able to run in real time due the fact that it uses the CPU for all the simulation processing.

Our framework has a simulation that runs on GPU instead of CPU to achieve better performance. It provides the user a fast, ease, and safe environment for researches using most of the technology that is used in cutting edge games. At the same time, it is very easy to expand due its modularity.

3 The Framework Architecture

The proposed architecture was built with easy use and extensibility in mind, adopting a plug-in strategy [Birsan 2005]. In this case, anyone who needs a functionality that is not present in the framework can add it without the necessity of modifying or even recompiling the framework, by only attaching the new developed module to it. In Figure 1 a simple class diagram of the entire framework is shown. As it can be seen, the core subsystems that compose it are the services, the game screens, and the game object subsystem. Each of them is presented in more detail below.

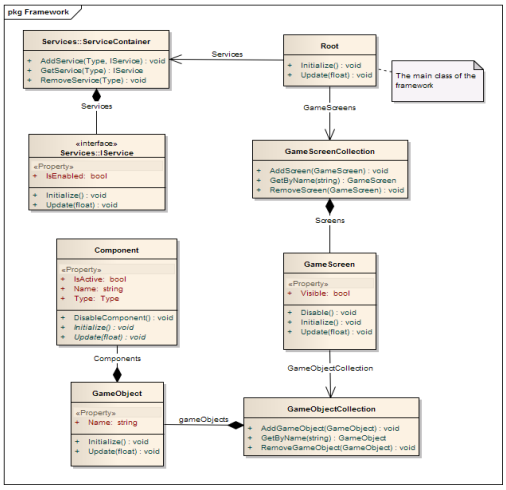


Figure 1: *The framework architecture*

3.1 Services

Services in the framework are modules that are available to be used entirely in all stages of the simulation. The basic concept is that services are unique and their tasks are well defined. Tasks that are strong candidates to be implemented as services are rendering, input event handling, and sound management. With this basic architecture of services, most of the references that a class needs to maintain are avoided, like a model that needs to use a render manager to be displayed. Due to its nature, the framework only allows the instantiation of one service per type.

As services are unique in the framework, problems of initialization can occur. One example of such problems is related to the physics service. Sometimes this class will need a service of rendering for debug purpose and probably will store a reference of this service inside of it. As services are created one after another, the physics service would not know if the rendering service is available at the moment of its initialization, requiring, at least, one boolean check to verify if the required classes have already been created. To avoid this problem, the services are initialized in two steps by the framework. In the first step, all services are created by the framework in a batch. In the second step, the framework calls the *Initialize* method of all services. At this moment, each service can initialize itself and asks for its services dependency in a safe way.

3.2 Game Screen

Game screens are used in the framework as a way to organize and manage a collection of scene elements. It can be thought as a collection of objects that are updated in batch by the engine, with some methods and properties to facilitate the organization and management of this collection in the framework.

Most physics libraries use some kind of container to organize all objects in the simulation that can interact with each other. In this case, objects that reside in a space have no influence over objects that resides in another space, making possible more than one simulation running in parallel. Using game screens it is possible to simulate the same behavior, as the framework runs them in parallel and objects that are in a game screen does not have influence over objects that are in another, although the same object can be added to more than one game screen. In order to facilitate the user, the framework already provides the derived class *PhysicsSimulationScreen*, which initializes and starts all required services necessary to work with physics simulation.

3.3 Game Objects

A game object is the basic and only element that can be simulated in the framework. It is based on the concept of Game Object Component System [Bilas 2002]. In this system, the author proposed the use of aggregation instead of inheritance to define a new functionality in a game object. Using this approach, some problems related to inheritance are avoided and new game objects can be easily implemented by the user, who only needs to define new components to handle the required specific task.

In order to be simulated, game objects need to be attached to a *GameScreen* object, which is responsible to update each game object in the right time during the simulation. In advance, to better manage game objects, they need to be unique in the game screen. To achieve this, each game object has an identifier, which is set by the user during its instantiation. This identifier is used to avoid the same game object to be added more than once in the same game screen, although it can be added in more than one game screen.

Game objects, to be properly used, are composed of a collection of components, as discussed before. Components in game objects may need to use other components in order to work appropriately. This kind of dependency is solved by using a technique called dependency injection [Passos et al. 2008].

4 Built-in Services

As discussed before, services are an important mechanism to enable extensibility of the framework. Although they can be created by the user, the framework provides some basic services to allow it to run. All the services listed below are necessary for allowing a physics simulation.

4.1 Input Service

Input services are responsible to deal with input devices. Currently, the framework has only support to deal with mouse and keyboard devices but others can be easily added by the user.

4.2 Graphics Service

Graphics tasks need to deal with different kinds of models that can be rendered differently from other kinds of models. One example of this is the shader effect that can vary from one model to another, requiring different parameters so that the model is properly rendered. As a consequence, each type of model could require the creation of a different class of render component in order to render the models of that type. Even more, *GameObject* is a sealed class, so it is not possible to simply extend this class to deal with custom parameters in the models.

The adopted approach to work with graphics is shown in Figure 2. As an example, the *XNAModel* class is shown in the diagram. It is a wrapper for the built-in *Model* class from XNA, which is responsible for rendering geometry. In this case, the interface *IRenderable* needs to be realized in order to use the built-in *Model* class for rendering. This interface realization is done by the *XNAModel* class, which encapsulates the built-in *Model* class. With this approach, a *Content Pipeline* extension, which processes data types defined by users, can be easily created by using a custom model class that realizes *IRenderable* interface and maybe inherits from the built-in *Model* class. The use of this wrapper was necessary because this class does not implement the *IRenderable* interface, which is required to render a model.

The next step is the setting of custom shader parameters. In order to do that, the *RenderableComponent* also needs a class that implements the *IEffectUpdater* interface. In the diagram, *BasicEffectUpdater* is a class that implements this interface. It also must inherits from *Component* class because *RenderableComponent* will find it in its parents' component collection, due to the dependency injection method.

With this restriction, a class that implements *IEffectUpdater* interface also needs to inherit from *Component* class. As the components have a parent class, the *IEffectUpdater* will have access to all data from a game object, allowing custom parameters to be setup in a shader.

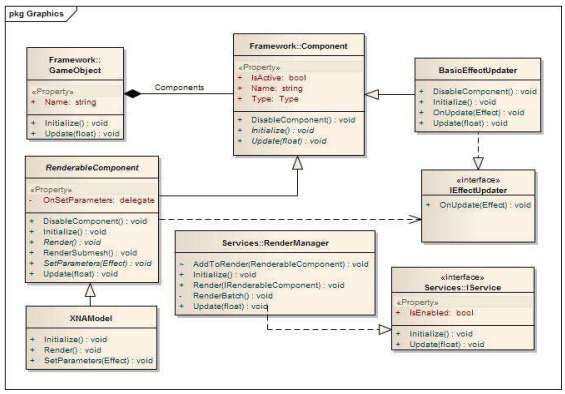


Figure 2: *The graphics service*

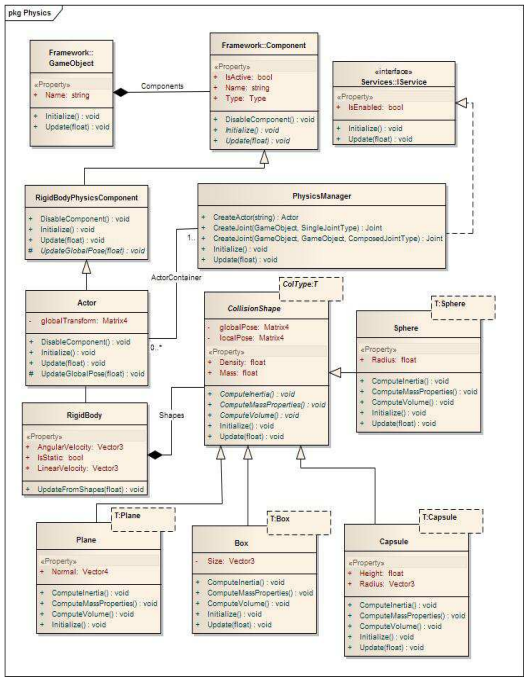


Figure 3: *The physics service architecture*

Finally, when it is time to render, the framework will invoke an internal method of the *RenderManager* service which will render all models that were added to its collection, making some optimization in this process.

4.3 Physics Service

This service is one of the most important in the framework, as it is responsible for the physical world simulation. This physics service was made using CUDA, which yields high performance physical world simulation in the framework, as physics tasks can be highly parallelized.

Following the overall framework architecture, for an object to be simulated using physics laws, a game object needs to be created and it must have a physics component, as can be shown in Figure 3. With this architecture, it is easy for the user to create physics objects in the framework, as the only requirement is setting a physic component to the game object to be simulated.

As an important feature, joints are also supported by the framework through the physics service which needs two game objects as a parameter. In this case, these game objects will react according to the joint constant added to them.

5 Performance and Analysis

In this work, we evaluated the framework against jMonkeyEngine [jMonkeyEngine 2003], a 3D engine that uses Java for simulation programming. We choose jMonkeyEngine because Java is an easy to use and powerful language for developing real time simulations. Additionally, Java is very comparable to C# as both use a Virtual Machine (VM) for running applications. For this comparison, we used jMEPhysics [JMEPhysics], an interface that connects a variety of physics library to jMonkeyEngine, allowing the user to simple use it without worries about integrations with the render engine.

At first, we planned to evaluate the framework against VPython as it is widely used in academic for physics simulations [Salgado 2001]. Unfortunately VPython has not a built in physics module or does not use anyone available. In this case, the integration needs to be done by the user if physics simulations are necessary.

The tests were performed on an Intel Core 2 Duo 2.4Ghz CPU, 4GB of RAM equipped with a NVidia 9800 GTS GPU. Each instance of the test ran for 30 seconds and the average time to compute a frame was recorded for each one. To assure the results are consistent, each test was repeated 5 times.

A total of 10 different test instances were performed for each framework type, varying only the number of balls, ranging from 50 to 16000.

From the raw results, shown in Table 1, it is possible to see the performance of the XNA framework using GPU for physics over a CPU physics based. From this table, it is possible to see that physics simulations in JMonkey decreases in a exponentially manner whereas it is more linearly in XNA. Using GPU physics based, more accurate simulations can be made as more powerful is given for researches, as well.

6 Conclusions

In this paper, a framework that enables real time physics applications with graphical feedback was presented. The main purpose of this framework is to help educators and researches in physics, without background in computer programming and graphics programming, to use technologies that are mainly used in games to simulate physical environments. This is achieved by using a solid architecture and a computer language that is easy to use and do not have collateral effects. Additionally, it uses one of the most promising approaches to simulate physics — the graphics processor unit and CUDA to parallelize the computations, releasing the CPU from these tasks.

The framework is a constant work in progress and we are now investigating how use the framework for simulating soft bodies and fluids to allow a wide range of physics simulation enabled in real time. We also plan to extend this to developing a visual editor to allow the simulation to be created visually, without requiring programming by the user of the framework. This could facilitate even more uses of the framework.

References

BILAS, S. 2002. A data-driven game object system. Game Developer Conference.

BIRSAN, D. 2005. On plug-ins and extensible architectures. *Queue 3*, 2, 40–46.

INTEL, 2000. Havok. http://www.havok.com/.

JMEPHYSICS. Jmephysics: interface between jme and physics engines. https://jmephysics.dev.java.net.

JMONKEYENGINE, 2003. Jmonkeyengine 2.0 on line documentation. http://www.jmonkeyengine.com.

KAČIĆ-ALESIĆ, Z., NORDENSTAM, M., AND BULLOCK, D. 2003. A practical dynamics system. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 7–16.

MICROSOFT, 2004. Creators club. http://creators.xna.com/en-US/.

MICROSOFT, 2008. Microsoft robotics. http://msdn.microsoft.com/en-us/robotics/default.aspx.

NVIDIA, 2004. Cuda zone. http://www.nvidia.com/object/cuda_home.html#.

NVIDIA, 2008. Physx. http://www.nvidia.com/object/physx_new.html.

PAPASTERGIOU, M. 2009. Digital game-based learning in high school computer science education: Impact on educational effectiveness and student motivation. *Comput. Educ. 52*, 1, 1–12.

PASSOS, E. B., SOUSA, J. W. S., NASCIMENTO, G., AND CLUA, E. W. G. 2008. Fast and safe prototyping of game objects with dependency injection. In *VII Brazilian Symposium on Computer Games and Digital Entertainment*, Sociedade Brasileira de Computao - SBC, 64–69.

POPOVIĆ, J., SEITZ, S. M., ERDMANN, M., POPOVIĆ, Z., AND WITKIN, A. 2000. Interactive manipulation of rigid body simulations. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 209–217.

PRICE, C. B. 2008. The usability of a commercial game physics engine to develop physics educational materials: An investigation. *Simul. Gaming 39*, 3, 319–337.

PUC-RIO, 1993. The programming language lua. http://www.lua.org/.

SALGADO, R., 2001. Vpython applications for teaching physics. http://www.phy.syr.edu/~salgado/software/vpython/.

SCHERER, D., 2000. Vpython. http://vpython.org/.

SHAPIRO, A., CHU, D., ALLEN, B., AND FALOUTSOS, P. 2007. A dynamic controller toolkit. In *Sandbox '07: Proceedings of the 2007 ACM SIGGRAPH symposium on Video games*, ACM, New York, NY, USA, 15–20.

STRONG, A., 2002. America's army official website. http://www.americasarmy.com/.

UNITY, 2004. Unity3d. http://www.unity3d.com.

WORKMAN, K., AND ELZER, S. 2009. Utilizing microsoft robotics studio in undergraduate robotics. *J. Comput. Small Coll. 24*, 3, 65–71.

YEH, T. Y., FALOUTSOS, P., AND REINMAN, G. 2006. Enabling real-time physics simulation in future interactive entertainment. In *Sandbox '06: Proceedings of the 2006 ACM SIGGRAPH symposium on Videogames*, ACM, New York, NY, USA, 71–81.

Table 1: *Comparison Tests*

# actors	time JMonkey	fps JMonkey	time XNA	fps XNA
50	1,67	598,41	4,25	235,29
1000	78,61	12,72	10,60	94,34
2000	144,92	6,90	18,43	54,26
4000	202,02	4,95	34,33	29,13
12000	-	-	148,20	6,75
16000	-	-	213,35	4,69

A Game Loop Architecture with Automatic Distribution of Tasks and Load Balancing between Processors

Marcelo Zamith
UFF-IC-MediaLab

Mark Joselli
UFF-IC-MediaLab

Luis Valente
PUC-RIO, ICAD Games

Esteban Clua
UFF-IC-MediaLab

Anselmo Montenegro
UFF-IC-MediaLab

Regina Celia P. Leal-Toledo
UFF-IC

Bruno Feijo
PUC-RIO, ICAD Games

Abstract

Nowadays, multithread architectures for PCs (multi-core CPUs and GPUs), and game consoles (as Microsoft Xbox360 and Sony Playstation 3) is a trend. Hence, single thread games loops will not get the best performance on such architectures. For this reason, multithread game loops that take advantage of such architectures are gaining importance. There are a lot of multithread game loops that can be used in order to achieve better performance in a game, but they can not be adapted for different architectures. This paper presents a new architecture for game loops that can detect and analyze the user hardware and adapts itself to a specific game loop that can achieve the best performance for that hardware.

Keywords:: Game loops, GPGPU, Task Distribution, Load Balancing, Real-time Systems

Author's Contact:

{mzamith,mjoselli,esteban,anselmo,leal}@ic.uff.br
bruno,lvalente@inf.puc-rio.br

1 Introduction

Multi-thread architectures on PC are getting more and more common with the development of multi-core processors and the new GPUs architectures that can be used for generic processing. Also top of the line video games like the Microsoft Xbox 360 and the Sony Playstation 3 features multi-cores processors. With that, game architectures have to paralyze and distribute its tasks between the processors, needing to utilize concepts from distributed and parallel systems in order to fully take advantage of the hardware.

Games are interactive real-time systems and, like multimedia application, they have time constraints to execute all of its processes and present to the end user the results. If a game does not fulfill this requirement, it will lose its interactivity and consequently it will fail. A common parameter for measuring a game and simulations is in frame per second (FPS). The lower acceptable bound for a game is 16 FPS. There are not higher bounds for a game FPS, but in PCs when the refresh rate of the monitor is less than the refresh of the game some discard of the rendered frame may occur. In order to achieve the best FPS in a game, game loops are designed and developed.

Game loop is the structure that determinate the order that each task of the game is executed during the loop. The game loops is mainly divided in three categories: data acquisition, which gets the data from user's input; data processing, where the game logic are processed; and data presentation, where the results are presented to the end user though images and audio.

This work is an extension of the work [Joselli et al. 2008], where a framework for game loops that can automatic distribute task between CPU and GPU and also can implement single or multi thread game loops. This work extended that work by presenting the following concepts: an automatic verification of the user hardware, as well as selects which loops will be used based on that knowledge and load balancing between the threads of the gameloop.

This work is organized as follows: Section two presents game loop background. Section three presents some related works. Section four presents and explains all the major functionalities of the proposed architecture. In section five presents the conclusion.

2 Game Loops

The game loop is the underlying structure upon games are built. Games are regarded as real-time applications because this kind of application has time constraints to run the tasks that rely on them. This means that if those tasks do not run fast enough, the experience the game must provide will be compromised.

The tasks that a computer game should execute can be broken down into three general groups: data acquisition, data processing, and presentation. Data acquisition means gathering data from available input devices: mice, joysticks, keyboards, and motion sensors. The data processing part refers to applying the user input into the game (user commands), applying game rules (game logic), simulating the game world (game Physics), simulating non-player characters (Artificial Intelligence), and related tasks. The presentation refers to providing feedback to the user about the current game state, through images and audio.

As listed previously, there are many tasks that a game must run. A computer game provides the illusion that everything is happening at once. Since a computer game is an interactive application, if it is unable to perform its work on time, the user experience will not be acceptable. This issue characterizes computer games as a heavy real time application.

3 Related Works

Although the game loop represents the heart of computer games, there are not many academic works devoted to this subject. The works by blind for review et al [Valente et al. 2005], Dalmau [Dalmau 2003], Dickinson [Dickinson 2001], Watte [Watte 2005], Gabb and Lake [Gabb and Lake 2005], and Mnkknen [Mnkknen 2006] are among the few ones. All of them focus on single-player games. The simplest real time game loop models are the coupled ones. The Simple Coupled Model [Valente et al. 2005] perhaps is the straightforward approach to modeling game loops. It consists of sequentially arranging the tasks in a main loop. Figure 1 depicts this model.

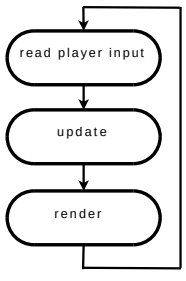


Figure 1: *Simple Coupled Model*

This first model runs as fast as the machine is able to, making it very unpredictable when it comes to using it in different machine configurations. The uncoupled models separate the rendering and update stages, so they can run independently, in theory. A nave approach to a real time uncoupled models is to use one thread for rendering and another for the update tasks. This approach exposes the same unpredictable behavior of the Simple Coupled Model. The Multi-thread Uncoupled Model [Valente et al. 2005] try to bring determinism to the game execution by feeding the update stage with a time parameter. Figures 2 illustrate these models, respectively.

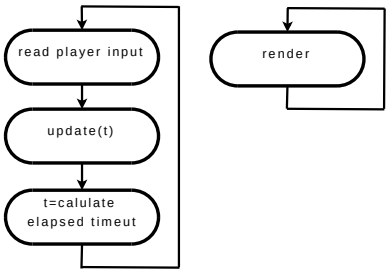


Figure 2: *Multi-thread Uncoupled Model*

By using these models, the application has a chance to adjust its execution with time, so the game can run the same way in different machines. More powerful machines will be able to run the game more smoothly, while less powerful ones will still be able to provide some experience to the user. Although these are working solutions, time measuring may vary greatly in different machines due to many reasons (such as process load), making it difficult do reproduce it faithfully. For example, some games may require a scene replay feature [Dickinson 2001], which may not be trivial to implement if it is not possible to run some game sequence in a deterministic way.

Nowadays, computers and new video game consoles (such as the Xbox 360 and the Playstation 3) feature multi-core processors. For this reason, game loops that take advantage of these resources are likely to become important in the near future. Therefore, parallelizing game tasks with multiple threads is a natural step. However, dealing with concurrent programming introduces another set of problems, such as data sharing, data synchronization, and deadlocks.

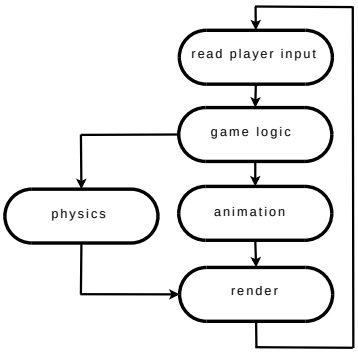


Figure 3: *Synchronous Function Parallel Model*

The author states that this model is limited by the amount of available processing cores, and the parallel task should have little dependency on each other. The Asynchronous Function Parallel Model [Mnkknen 2006] is the formalization of the idea found in [Gabb and Lake 2005]. This model does not present a main game loop. Figure 4 illustrates the model.

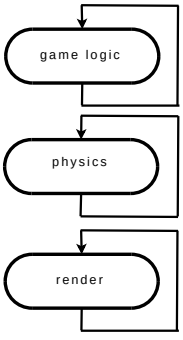


Figure 4: *Asynchronous Function Parallel Model*

Different threads run the game tasks by themselves. The model is categorized as asynchronous because the tasks do not wait for the completion of other ones to perform their job. Instead, the tasks use the latest computed result to continue processing. For example, the rendering task would use the latest completed physics information to draw game objects. This measure decreases the dependency among tasks. However, task execution should be carefully scheduled for this scheme to work nicely. Unfortunately, this is often out of the scope of the application. Also, serial parts of the application (like rendering) may limit the performance of parallel tasks [Gabb and Lake 2005].

According to [Mnkknen 2006] , this model scales well because it is able to allocate many processing cores as there are available. Performance is limited by the amount of data processing that can run in parallel. An important issue is how to synchronize communication of objects running in different threads. Mnkknen 2006 states that the biggest drawback of this model is the need to having components designed with data parallelism in mind.

Josseli et al [Joselli et al. 2008] presents an architecture for game loops is able to implement any game loop model and distribute tasks between the CPU and the GPU. This work extends the work by Blind for review by proposing a game loop that is able to detect the available hardware and automatically distribute tasks among the various CPU cores and also to the GPU.

4 The Proposed Architecture

Although processing power in consoles and computers has greatly increased, and multi-core architectures make it possible to use parallel processing, proper software is needed to extract high performance from the hardware. Even though the game loop concept applies to both consoles and computers, there are some differences among these kind of hardware.

Consoles (i.e. consoles in the same family such as the Xbox 360) have the same hardware, memory, processors and number of cores, making development in those platforms more predictable (i.e. the developers knows the hardware s/he will be working with). On the other hand, for computers there is a myriad of configurations considering processors, memory, GPUs, and combination of this (and other) hardware.

The proposed architecture works with multi-core CPUs and GPUs (if one is available). The architecture considers both as resources. A resource is a CPU core or a GPU, and the architecture encapsulates them. However, not all of game tasks are suitable for processing both in the GPU and the CPU, as their architectures are different and require different programming paradigms.

The aim of the proposed architecture is to provide a management layer that it is able to analyze dynamically the hardware performance and adjust the amount of tasks to be processed by the resources. In order to make a correct task distribution, it is necessary to run an algorithm, and in the current architecture, a script is responsible for this. The architecture applies the scripting approach because the game loop is used in many games, and for each of them it uses a different algorithm and a subset of its parameters.

The core of the proposed architecture corresponds to the Task Manager and the Hardware Check class. The Task Manager schedules tasks in threads and changes which processor handles them whenever it is necessary. The Hardware Check monitors the processors to calculate the usage rates of all of them.

4.1 The Task Class

A task is defined such as a work that the application (the game) should execute. There are three classic tasks that all game application execute: player input, game state update, and providing feedback to the player. These tasks should be broken in small subtasks, for example: Player input is performed by reading the keyboard, mouse or joystick state. Feedback to the player is provided by rendering the scene graphically, playing a sound effect, or vibrating the joystick. Hence, a task can be anything that the application works towards processing.

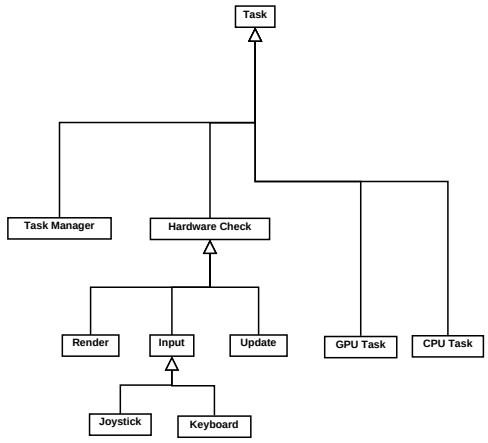


Figure 5: *UML class diagram*

A constraint of this architecture is the fact that not all tasks can be processed by all sorts of processor architectures. An example is the CPU and the GPU, that have different hardware architectures. The GPU hardware is not able to access all of computer peripherals just like the CPU. For example: read a file from hard disk. It is a task that only the CPU is able to process. On the other hand, physics simulations can be processed by both, although GPUs typically execute it faster than CPUs.

Thus, a task is broken in three groups. The first group works with tasks that can be invoked only by the CPU in its cores, such as player input, file handling, and the managing of other tasks. The tasks of this group can be allocated in different cores, and, if there are more than CPU core, they can run in parallel. In the same way, the second group handles the tasks that can be only processed by GPUs, such as running shader processing. The last one is the group that can be modeled to be invoked by both processors. Figure 5 illustrates the UML class diagram for tasks.

The third group is the main focus of this work, because its tasks can be invoked by all processors. We are specifically interested in the issue of delegating a task to a CPU core or to the GPU.

The task structure is described as follows:

- TASK_ID: it is the id of the task. Whenever a task is created, it receives this unique number. The architecture uses the id to guarantee the correct order of execution and to identify which task is accessing the data shared among the tasks;
- TASKTYPE: the task type, that can be the following: input, update, presentation, and management (this last one identifies Task Manager and the Hardware Check classes);
- STACK: It is a stack area used to preserve the task state whenever it leaves the processor;
- DEPENDENCY: ID of next task that should run after this one. The Task Manager relies on this information to guarantee the correct execution ordering;
- PRIORITY: The Task Manager uses this value to decide which tasks in the task queue are going to be scheduled in each processor. The next Section provides more details about this field.
- Load balancing between the threads of the work.

4.2 The Task Manager Class

The Task Manager is a special class derived from the generic Task class, as Figure 5 illustrates. It provides management, instancing, finalizing, and synchronization of all others tasks (except itself).

To guarantee the correct distribution and the abstraction of the tasks, the architecture defines a special task called Task Manager. The Task Manager is responsible for defining which task runs in which processor, and it should check whether a task can be broken in smaller tasks or not, and redistribute them. This class is derived from the generic Task class, and it provides management, instancing, finalizing, and synchronization of all others tasks (except itself). In order to guarantee central control of tasks and their correct execution, there is only one instance of the Task Manager class in the application, implemented as the Singleton design pattern.

The Task Manager class loads scripts that describe the initial policy of distribution that should be adopted. This means which messages it will send and the rules that should be applied. The input parameters of the script are: the task name, the elapsed time, and which processor is being used. The script has a task list and their constraints. The task constraints are: the execution ordering and which processor should be used.

For each kind of task, the Task Manager holds one version of the algorithm for the CPU (CPU Task) and another version (if applicable) for the GPU (GPU Task), both of them with the same distribution policy. The Task Manager creates instances of the GPU Task objects to run in the GPU thread. If there is a processor in the system with four cores, the Task Manager will create four instances of the CPU Task class, according to the script rules. Thread synchronization must be guaranteed in the algorithms themselves. The GPU Task objects work the same way. The Task Manager identifies how many GPUs are there in the system, and then it breaks the task among them.

To make it possible to use a parallel programming model, the Task Manager implements a multithread game loop. To avoid the problems of this model of programming, the Task Manager uses a semaphore such as the synchronize object. Accessing shared data, starvation and deadlocks are examples of parallel programming model problems. Figure 6 illustrates the game loop model this work proposes.

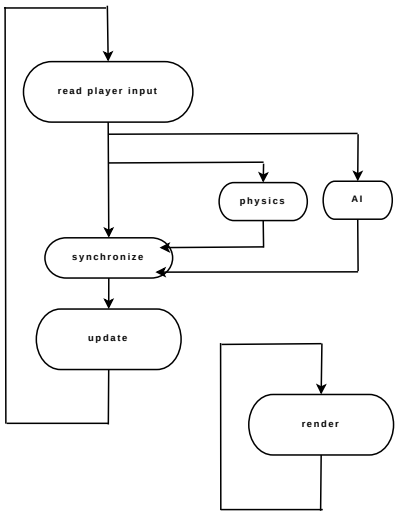


Figure 6: *Multi-thread Uncoupled Model (CPU/GPU)*

The Task Manager sees the CPU cores and the GPU as processing resources. To distribute best the application among these resources, the Task Manager exchanges messages with the Hardware Check objects (they are described in the next Section). These messages are exchanged during the execution of the application. The Hardware Check objects send information about the performance of the tasks to the Task Manager, which in turn uses this to decide about the best load balancing to apply.

Each task holds information that the Task Manager queries and updates. Recapping from Section 4.1, these information are: task id, task type, a stack, dependencies on other tasks, and task priority.

The tasks send messages to the Task Manager when these events occur:

- A task finishes execution;
- A task finishes processing shared data.

When a task finishes processing some shared data, the message it sends to the Task Manager is interpreted as a release of the shared data.

For each CPU core and each GPU, there is a processing thread and an associated task queue. When a task finishes executing, the Task Manager chooses the next task to run based on the priority values of the tasks in the queue.

One of the major features of the proposed architecture is scheduling a task to run on another processor (CPU core to GPU or GPU to CPU core or CPU core to other CPU core) during its execution. In these cases, the task state is pushed to the tasks own stack (and later restored) regardless of the processor type. For example, in time t_1 the GPU processes a Physics task and in time t_2 this task is scheduled to the CPU. When the task starts to run again (now in the CPU), the Task Manager reloads the task state from the tasks stack and signals it that the processor type has changed. The task priority is changed to a value of zero, which means that the task is placed on the front of the task queue. This measure is a way to guarantee that the task will keep on running.

The Task Manager performs load balancing according to the usage rate of processors. Another class, named Hardware Check, is responsible for providing the information about usage rates. Section 4.3 describes this class.

4.3 The Hardware Check Class

The Hardware Check is implemented as a task that runs on the CPU. There is only one instance of this class in the application. This class keeps track of the number of CPU cores and GPUs available in the system. While the application is running, it watches each processor to calculate the usage rates. The class informs the Task Manager about the current state of the processors and their cores. The possible states are: ”waiting” and ”running”.

The class uses an ordered data structure to store information about the usage rates of each processor. In this case, using a simple vector is enough. Keeping the vector ordered guarantees that the first element will be that represents the highest usage rate.

4.4 Ways to Distribute the Tasks

The architecture performs the initial task distribution based on heuristics by reading a script file that contains the rules. This script file is written with the Lua language [Lua 2009]. This script should not be complex to avoid delaying application startup.

The distribution of the tasks is done based on heuristics described and load before the execution of the application and it is written in LUA language . Despite the script of the heuristic is being loaded and processed during the start of application, it should not be complex to avoid spending time processing.

The architecture supports two kinds of heuristics. The first one is manual, which means specifying tasks for specify processor, ignoring performance. This heuristic is used to test task performance. The second one is the automatic heuristic.

The automatic heuristic should describe all tasks used by the application, their ordering and the types of processors the tasks use, as well as the rules to apply for changing processors. The elapsed time or frames per second are the only information the heuristic uses. It is a simple measure and can be applied for GPUs and CPUs. The input parameters are the processor ID and the time elapsed by the processor to process it. It returns the processor ID for the processor that should run the task after that execution.

The architecture automatically distributes tasks between processor cores. If a core takes longer to process a task than other cores, the Task Manager removes some of the load of the heaviest task, the Task Manager removes some of the load of the heaviest task (i.e the task that is taking longer to run), and put in the lightest task (i.e the task that is taking less time to run).

5 Conclusion

The development and evolution of multi-cores processors, GPUs and video games indicates that multithread architectures is a trend.

This work discussed the concept of game loops, a subject that is not very discussed in the literature. Our contribution lies on extending a previous work by providing an architecture for game loops that is able to distribute tasks between the CPU and the GPU.

The proposed architecture is able to detect the available hardware, and then to break tasks into CPU cores and, if it is available, send them to the GPU.

References

DALMAU, D. S. C. 2003. *Core Techniques and Algorithms in Game Programming*. New Riders Publishing.

DICKINSON, P., 2001. Instant replay: Building a game engine with reproducible behavior. Available at `http://www.gamasutra.com/features/20010713/dickinson_01.htm/` .

GABB, H., AND LAKE, A., 2005. Threading 3d game engine basics. Available at `http://www.gamasutra.com/features/20051117/gabb_01.shtml/` .

JOSELLI, M., ZAMITH, M., VALENTE, L., CLUA, E. W. G., MONTENEGRO, A., CONCI, A., FEIJO, B., DORNELLAS, M., LEAL, R., AND POZZER, C. 2008. Automatic dynamic task distribution between cpu and gpu for real-time systems. *IEEE Proceedings of the 11th International Conference on Computational Science and Engineering*, 48–55.

LUA, 2009. The programming language lua. Avalible at: `http://www.lua.org`. 13/01/2009.

MNKKNEN, V., 2006. Multithreaded game engine architectures. Available at `http://www.gamasutra.com/features/20060906/monkkonen_01.shtml` .

VALENTE, L., CONCI, A., AND FEIJO, 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.

WATTE, J., 2005. Canonical game loop. Available at `www.mindcontrol.org/~hplus/graphics/game_loop.html/` .

A Method for the Use of Patterns Drawn from Cellular Automata as Generative Source for Discrete Virtual Environments

Gottin, V. M. Schardong, G.G. Felipetto, C.M. Pozzer, C.T.
Universidade Federal de Santa Maria, LaCA – UFSM, Brazil

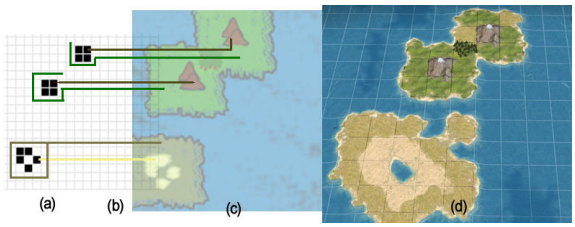


Figure 1. A graphical representation of the method proposed. The patterns in a cellular automaton (a) are translated through a set of rules (b) into generative data structures, here depicted graphically (c) containing information for a rendering process to generate an environment (d).

Abstract

This paper proposes a method for creating discrete virtual environments using recognition of patterns from Cellular Automata. The application of rules over the recognized patterns generates the data structures describing the virtual environment. The steps for implementation are described and a case study is presented. Results are analyzed and future works for harnessing and developing the method are proposed.

Keywords: Cellular Automata, environment generation, discrete environments;

Author's contact: {vmgottin, schardon, felipeto, pozzer} @ inf.ufsm.br

1. Introduction

Cellular Automata (CA) have been widely explored due to their intrinsic characteristic of simulating complex behaviors parting from sets of simple rules [Wolfram 2002]. This paper proposes a method for the creation of environments through a composition of discrete components derived from patterns in a cellular automaton.

An application of the proposed method consists of the translation of recognized patterns in a cellular automaton's configuration into environment components aggregated into a single environment. As an application based on CA, it might benefit from some characteristics that enable performance gain.

Additionally, this paper explores the possibility of creating environments capable of 'evolution' following the logical rules of the cellular automaton it is based on. Hence, this paper brings two original contributions: a new procedural generation method for creation of content for tile-based games and a new proposal for simulation of topographic evolution.

2. Related Work

CA have been widely studied and much remains to be explored. The interpretation of patterns in CA is a field of study all by itself, being the subject of many studies [Chou 1996]. Use of CA for systems simulation is also the subject of many researches [Lima 2005], for physical systems containing many discrete elements with local interactions are often conveniently modeled as CA [Wolfram 1983].

The proposed method aims to generate discrete environments through the creation of aggregated discrete environment components, thus fitting into that description. Not much has been specifically done about the creation of virtual environments utilizing CA, save for very specific components for non-discrete environments, like the approach presented by Judice et al. [2008].

3. Cellular Automata

CA are models of complex systems in which an infinite lattice of finite state machines updates itself in parallel according to an isotropic local rule [Culik et al. 1990]. Development of methods, techniques and algorithms relating CA to other areas [Wolfram 2002] are a result of the interest and possibilities over the characteristics of CA: complex behavior through the application of simple rules [von Neumann 1966]; and the possibility of viewing the evolution of a system like a cellular automaton as a computation. Sipper [1999] gives various examples of applications using CA that show the three characteristics of this paradigm: simplicity (each cell can do very little work separately, leaving a very narrow margin for error), vast parallelism (vast numbers of cells operating in parallel) and locality (connections between cells are local).

Thus, it is proposed that the parallelism and the possibility of performance enhancement make applications based on CA a viable alternative to pre or randomly generated environments. Nevertheless, specific statistical research over the proposed method is needed.

4. The *Harp* Method

The objectives and application of *Harp* significantly differ from other techniques of virtual environment generation [Mandelbrot 1982; Perlin 1985] that are somewhat focused on real-time processing of content. While these techniques aim to render real-like terrain, *Harp* focuses on creating data structures over which an environment may be rendered. The rendering step is also not a part of *Harp*. In the case study, Civilization IV's engine was used for the rendering of a discrete environment.

The choice over a particular cellular automaton for a fine-tuned implementation of *Harp* must take into consideration the dimension, rules, cellular states and

structure permanence of the automaton, as in the classes defined by Wolfram [2002]. Parameters for this choice would require further research, not in the scope of this proposal and, as such, throughout the article all examples – including the case study - of the application of *Harp* will be presented utilizing a *Conway's Life* [Gardner 1970] implementation, one of the most extensively explored CA[1].

The first step of *Harp* depends entirely on the chosen cellular automaton. The step is very simple, consisting on capturing the configuration of this automaton and outputting it for Step Two: recognition of patterns in a given generation of the chosen cellular automaton. The implementation of the pattern recognition algorithm used in the case study is formalized as follows: given a space ***D*** of any dimension and a pattern with a set of live cells $\boldsymbol{L}\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_i, \ldots, \mathbf{x}_n\}$, with $\boldsymbol{n}$ live cells and $\mathbf{x}_i$ being the position in ***D*** of the ***i***-th live cell in the set; then the algorithm will recognize this pattern in ***D*** if:

- A set of live cells ***L'*** exist in ***D*** and;
- For every live cell ***i'*** in ***L'*** with a position $\boldsymbol{x}_{i'}$ in ***D*** there is a live cell ***i*** in ***L*** with a position $\mathbf{x}_i$ equal to $\mathbf{x}_{i'}$;
- The pattern configured by ***L*** presents the same immediate neighborhood of dead cells as ***L'***.

In the case of the possibilities explored above, the pattern recognition algorithm returns as a result a string of patterns with some information attached: its type, its position ***X*** (i.e, the set of positions ***x*** of all live cells and immediate neighborhood dead cells) on ***D*** and, through simple interpretation of its type, the number of live cells ***n*** composing it. This will, for the rest of this paper, be referenced as the *generative information* of a pattern.

The third and fourth steps in *Harp* are tightly connected. Their objective is to apply a set of transformation rules over a set of generative information, derived from the pattern recognition in the previous step, to obtain environment components.

In the formalization of Step Three, a single rule is described as ***R*** part of a set ***S*** of rules, and each input item ***p*** is part of the set including all input ***P.*** Every item in ***P*** has a type $\boldsymbol{p}_\mathbf{j}$, with **j** being an element descriptive of its nature. As seen in Figure 2, each rule ***R*** in ***S*** is an effect **f** (Figure 2.a) over a single type of input item **j** – in this case, the type is correspondent to a pattern type, a block (Figure 2.b). Here, the algorithm for environment components creation will compare each rule in ***S*** with each item in ***P*** (Figure 2.c) and, whenever the type **j** of **f** is identical to $\boldsymbol{p}_j$, the rule (the effect) will be applied over the item ***p*** (Figure 2.e expresses the output of this step).

Formally, a rule in a transformation method may be expressed as the mathematical model of strings and sequences of symbols as used in the theory of finite-state automata [Roche and Schabes 1997]. In this formalization, the input information of *Harp*, a sequence of patterns, forms a string. Each symbol composing a string, in this case, is one input item of Step Three – a pattern and the information it contains, as seen in the previous section. The alphabet over which such strings can be built is, in this case, derived from the patterns recognized by the method in Step Two.

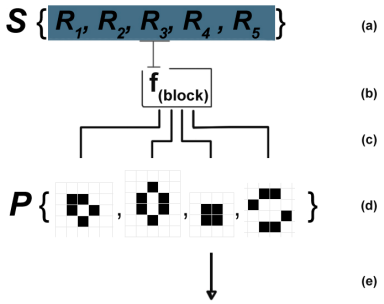


Figure 2. Each rule corresponds to a single effect over a single type of item. Each rule (i.e, each effect) is compared to each input item (here depicted as the graphical patterns in ***P***) and if their types are a match, than the effect **f** is applied in Step Four.

Three conclusions may be drawn from this formalization. First we additionally define, now within the formal model proposed, that every rule ***R*** in ***S*** must have an effect **f** related to every input item in the *alphabet*, and not in the input set ***P*** - i.e, one effect **f** related to every symbol in the alphabet. Second, the formal notation allows the creation of a simple standard for the notation of these rules – now considered symbols in an alphabet or characters in a string – which will be addressed later. Finally, as a last consequence of this formalization, it may be possible to implement operations like union, intersection, subtraction and inversion on a string that is a set of rules.

Step Four comprises of the creation of the data structures that will be rendered into a virtual environment for any kind of visual output. The output in the example would be a data structure with enough information to describe a grid with determined colors in determined spots. This information will be denominated *generative data*, in order to differentiate it from the *generative information*.

Step Four depends in great measure of the desired characteristics of the environment: its dimension, the level of detail, the rate of changes necessary for the environments simulation and anything else that the designer(s) consider necessary. This will be henceforth referred as the *design criteria*. First, though, an example of the output of Step Three, input to Step Four, is shown in Figure 3.

It is noteworthy that according to the rules of transformation applied in Step Three, effects are applied over the generative information, and that may not include its position in the dimension of the cellular automaton. In the example, the number of live cells in

[1] [online] Conway's Game of Life at the Open Directory Project: http://www.dmoz.org/Computers/Artificial_Life/Cellular_Automata/Conway%27s_Game_of_Life/

the pattern is used to determine the position of the components created.

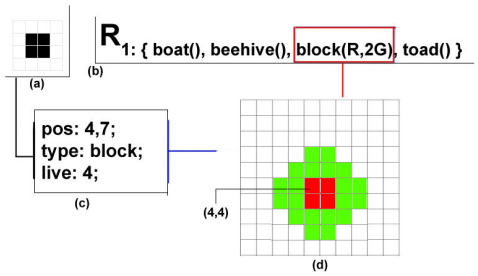


Figure 3 In this example, the pattern in the cellular automaton (a) translated through a rule (b) that changes its information (c) as in the previous example. The position of the component in the virtual environment generated is given by the coordinate of (***n***,***n***), with ***n*** being the number of live cells in the pattern. The position of the pattern in the automaton is unused information.

In conclusion, Step Four will output a generative data structure that contains information for a rendering technique to act. This rendering technique is not part of *Harp,* and in the following case study, a commercial game's scenario editor was utilized.

5. Case Study

In this section, a case study of *Harp*'s implementation is exposed. This study's design criteria is to test the application of *Harp* exposing in detail the results of each step of the method, meanwhile generating an environment with some coherence. The Civilization IV game engine was responsible for rendering the information in visual output. The approach used for this was to save the generative data outputted from *Harp* into a file readable by the engine. This was done through a C++ implementation intertwined with the *Life* implementation mentioned earlier.

Some adjustments to meet design criteria were experienced as part of the study. First, the set of rules was extended so that each pattern recognizes nearby patterns and include that information as part of its generative information. That way, not all generative data descendant from blocks in the automaton is the same. Without this, all landmasses generated from the same pattern would be identical. An example of this is shown in Figure 4.

Figure 4. An application of *Harp* where the generative information is based solely on the position of the pattern in the automaton generates several identical islands (a). When each pattern is affected by nearby patterns, more detailed and differentiated landmasses are generated, such as the one depicted in (b).

Second, a set of more composite rules - in which terrain features like hills were generated by the overlapping of patterns – was then developed to conform the output of *Harp* to the design criteria. An example is shown in Figure 5.

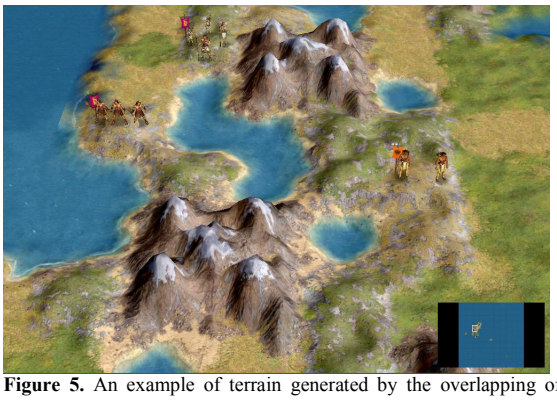


Figure 5. An example of terrain generated by the overlapping of generative data. Here, the hills were created as a result of the overlapping of block and blinker pattern's generative data.

Third, the use of multiple levels of *Harp* in the same environment was part of the experiment. Figure 6 shows a graphical representation of the concept, where a second cellular automaton (possibly with a new set of rules) has its patterns applied over a pre-existing environment – which was also generated with *Harp*. Figure 7 shows the results of utilizing a stance of *Life* to generate forests and jungle over the terrain depicted in Figure 5. This was accomplished by operating an intersection over the generative data: a forest was created in the environment where a landmass was created by the first stance *and* a forest was created by the second stance. This was possible to express in the chosen syntax because the sets of rules were formalized as strings.

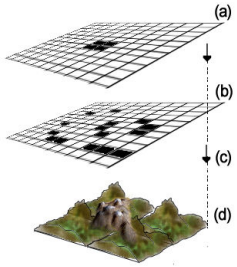


Figure 6. A representation of how a cellular automaton (a) outputs the data structures of its *Harp* application not for the rendering process, but for a second cellular automaton (b) that outputs for rendering (c) the information of the final terrain with multiple levels of detail (d).

Figure 7. In this example the landmasses are generated by one configuration of *Life* and the forests are generated by a second-level application of *Harp* with its own set of rules of transformation.

It was verified in these examples that yet more levels of application of *Harp* could be easily implemented, and that a concept of areas of interest might apply - i.e., the effects of a second-level automaton applying only over a desired part of the output of the first-level. A complete map with two second-level automata – one generating forests and the other, resources – acting over the landmasses produced by the first-level automaton is shown in Figure 8.

Figure 8. Screenshot of a map generated by the *Harp* method, using two second-level stances of *Harp* in addition to the first one that generates the landmasses.

A further objective of *Harp* is to develop the environment through the evolution (advancement of generation) of the CA over which it is based – basically, applying *Harp* iteratively. Since it is possible to predict the next state of the environment through the prediction of the automaton's next generation, it is possible to make this changes gradually appear in the environment, but this was not implemented and stands as the objective of current research. An example where the migration of forests in the scenario was created by subsequent generations of the same CA is shown in Figure 9.

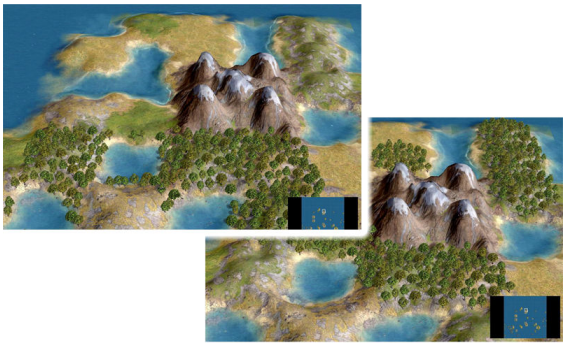


Figure 9. Multiple levels of *Harp*. The first level creates the landmasses and a second level creates the forests through a *intersection* operation of the sets of rules.

6. Conclusions

In this paper two original contributions are presented: first, a new approach for the creation of content for tile-based games through the method for generation of environments called *Harp* - and, secondly, indication that this method might emulate scenario evolution in a credible manner if proper implementation needs are met.

The case study presented showed some characteristics of the method. The design criteria for this study were fully satisfied. Some proposed techniques for *Harp*'s implementation were successful as well: local influence of patterns over each other; fine adjustment of rules for better results; multiple levels of CA generating the same environment and, finally, evolution of the environment through changes in the automata.

Some ideas were not explored and remain as focus for future research. First and foremost, the effect of the chosen automata's characteristics over the final result of *Harp* and parameters for the choice of the CA for meeting the design criteria must be researched. As exposed throughout the paper, this choice affects the entire application, and therefore, should be a priority in future works.

Manipulation of rules through operations on sets of rules expressed as strings, like the *intersection* operation between two sets of rule exemplified in the case study, are also an interesting field for future work. Operations between rules of the same set have not been covered by this study and should be a natural extension of the method.

References

CIVILIZATION IV, copyright 2005 Take-Two Interactive Software. http://www.2kgames.com/civ4/

CHOU, H., 1996. *Self-Replicating Structures in a CA Space*. University of Maryland, USA.

CULIK II, K., HURD, L. P. AND YU, S., 1990. Computation Theoretic Aspects of CA. *In: CA: Theory and Experiment*, H. Gutowitz, 1991.

GARDNER, M. 1970. Mathematical Games – The Fantastic combinations of John Conway's new solitaire game "life".

JUDICE, S. F., COUTINHO, B. B. S., GIRALDI, G. A., 2008. A CA Framework for Real Time Fluid Animation. National Laboratory for Scientific Computing.

LIMA, E. B., 2007. *Modelos microscópicos para simulação do tráfego baseados em autômatos celulares*. Doctorate thesis, Universidade Federal Fluminense.

MANDELBROT, B. B., 1982. The Fractal Geometry of Nature. W. H. Freeman and Company, New York, USA.

PERLIN, K. An Image Sunthesizer. Courant Institute of Mathematical Sciences, New York University.

ROCHE, E., YVES, S. 1997. Finite-State Language Processing. MIT Press: Cambridge, Massachusetts, USA.

SIPPER, M., 1999. The emergence of cellular computing. IEEE computer, v. 32, n. 7, p 18-26.

VON NEUMANN, J. *Theory of Self-Reproducing Automata*. University of Illinois Press, Illinois, 1966.

WOLFRAM, S., 1983. *Statistical Mechanics of CA*. The Institute for Advanced Study, Princeton, USA.

WOLFRAM, S., 2002. A New Kind of Science. Champaign: Wolfram Media.

A Proposal of a Shadow Detection Architecture Interface for Games Using Portable Projectors

Paulo M. F. Andrade
Media Lab – UFF

Micheli K. L. Andrade
Media Lab – UFF

Marcos Ramos
Media Lab – UFF

Esteban W. G. Clua
Media Lab – UFF

Aura Conci
Visual Lab – UFF

Anselmo Montenegro
Media Lab – UFF

Figure 1: The figures above illustrate: the game Shadow Volleyball (first figure, left), the use of a "pocket projector" connected to a mobile phone, the Sansung W7900 mobile projector phone and the projector and camera setup used during the tests.

Abstract

With the advent of "Pocket Projectors" - projectors based on LED [Rojas 2004], OLED [Miller 2009] or laser light emission [Captain 2008], with sizes similar to smartphones and the emergence of mobile phones capable of video projection, the "projector phones" [Ricker 2008], a new interface and a new way to play games became possible. By using computer vision techniques, the shadow generated by the human interaction with images projected by portable projectors can be used to control game elements and respond to game events. The player can interact with the projection using his shadow and the game interface can react accordingly.

This article proposes a interface for game control that react to commands based on the detection of shadows generated by players interacting with a projection from a portable or a common projector. This article also describes experiments made during the development of prototypes that respond to interactions represented by shadows.

Keywords: Computer Vision, User Interface, Computer Games, Mobile Phones, Game Development, Game Frameworks.

Authors' contact:
paulo@andrade.com
{micheli.knechtel,msramosjr}@gmail.com
{esteban,aconci,anselmo}@ic.uff.br

1. Introduction

In the video game world, two aspects of the market have become extremely relevant in recent years: games based on controllers that identify movement; concept enshrined by the Nintendo Wii, who will face a strong competitor with Microsoft's Project Natal [Microsoft 2009], and the popularization of mobile phones with color screens, making room for the massive growth in numbers of games for mobile phones. Both phenomena of portability and interaction caused a revolution in the game market, making room for more casual players and for games that need physical interaction.

Games for handheld devices and games that can be controlled by hand and body movement are popular, but until recently, they were a self-exclusion phenomenon; games with control based on body movement are not practical for portable devices and games for portable devices usually does not work well with body movements. Recently, the incorporation of accelerometer chips in mobile phones gave the ability to control games moving the phone. These movements are still limited to small turns and shakes since the player needs to see the screen to play properly. The advent of "Pocket Projectors" can change this barriers. With mobile phones connected to projectors, there is no need to look at the screen of the device, the player can look at its projection. However, while there is the possibility of wireless transmission from the mobile phone to the projector, little can be done for the detection of full or broad body movements of the player holding the phone for effective control.

With this scenario, the authors propose a new and unique paradigm of game interface using information captured from cameras of mobile phones, laptop cameras or USB cameras. This interface uses the shadow projected by the player over the image projected by the projector. This mechanism of interaction has several advantages over the current and future systems that detect motion based on analysis of body movement [Microsoft 2009]. Among the

advantages, we can mention the possibility of multiplayer games with movements like jumps and crouches, which is possible when interacting with projections of more than two meters in height. The size of the projection can actually influence in the mechanic of games allowing more physically challenging games.

This paper is structured as follows: the second section describes related works. The third section presents the computer vision techniques employed. The fourth section describes how the physical design of the play environment can influence the gameplay. The fifth section details the shadow identification and its influence in gameplay. The sixth section presents some game concepts using shadows and details how the shadow detection can be used in the gameplay of these games. The last section concludes presenting opportunities for expanding the research, along with some final thoughts and considerations.

2. Related Works

The use of image processing to infer information from shadows is not new. The U.S. patent number 6624833 [Kumar 2003] proposes an interface where the shadow of gestures is used to estimate the relative distance of the hand of the controller to a surface, in order to provide spatial information of the position of the hand. The proposed method also considers the identification of gestures by the interface.

The presence of shadows and the increase and decrease in size of the shadow according to the user's movement, considering the focus of the light, is used by Shoemaker et al. [2007] for user interaction with large screens, too big to be reached by the user touch. Another work that use shadows and big screens, now interactive big screens, is the work described in [Apperley 2002], where the system uses the silhouette of each user interacting in a virtual meeting, where several users simultaneously use big interactive boards or screens. The article describe the use of virtual shadows projected behind the content handled by each participant, in order to allow participants to know the relative virtual position of each other, despite being geographically distant and using their on boards.

Finally, another work that deserves mention concerns the patent number 20070300182 [Bilow 2007], that uses the shadow and the contact point of an object (such as the finger of a user) on the surface of a touch sensitive display to allow positioning and orientation of the interface elements.

3. Implementation and Computer Vision Techniques

The proposed interface use computer vision techniques to obtain information of the environment to allow the game loop to handle user interaction based on the requirements of the game. Since the information of the environment is produced by the projector and captured by the camera, the correct placement of both influence in both the gameplay and the information captured.

During the tests, the best placement is presented in Figure 2. Is important to emphasize that there is no need for a perfect alignment between the projection and the camera's field of view.

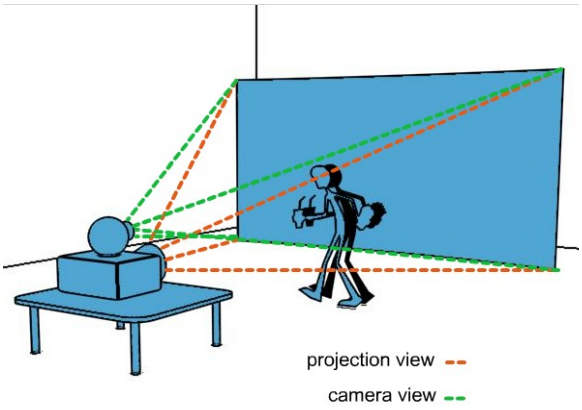


Figure 2: Camera and Projector Placement.

With the projector and camera positioned, the calibration process is started, which defines the region of interest of the game based on the color information of a rectangle generated by the projector.

To help identify the color, the RGB images acquired by the camera are converted to the HSV [Conci 2008] color space. The HSV model is a color system that helps the identification of a particular color and its variations in tone, which is very suitable for our purpose, since even if the background color is a pure, the light of the environment can still affect the information captured by the camera.

The shadow identification is done only in the region defined in the first step. Since just the shadow identification is necessary, the captured frames are converted do grayscale. After the conversion, the region is binarized, with a threshold low enough to detect only shadows. This step is necessary to address a phenomenon arising from the variation of illumination: noise, or more specifically, the false positives that can appear during the shadow detection.

In order to remove the noise, morphological erosion [Serra 1983] is applied. Only the pixel groups that contain the structural elements stay in the image. The groups that do not contain the elements are turned into black. After this, the image is prepared to be analyzed in order to calculate the bounding boxes of the detected shadows that will be used to calculate collisions.

The bounding boxes obtained are represented by Cartesian coordinates of the captured image region.

Hence, to make the collision calculations according to the actual projected image, it is necessary to transform the image of the captured Cartesian space to the Cartesian space of the game. To do that, the following equations are used:

X_screen=(screen_width/capture_width)*(x_capture–x_capture_min) (1)

Y_screen=(screen_height/capture_height)*(y_capture–y_capture_min) (2)

Where x_capture and y_capture represent the x and y coordinates of a pixel captured by the camera; x_capture_min and y_capture_min represent the coordinates of the minimum active area of the game; screen_width and screen_height represent the width and height of the screen of the game to be projected; capture_width and capture_height represent the width and height of the active region of the game captured; x_screen and y_screen represent the coordinates x and y of the pixel already transformed to the coordinates of the game screen to be projected.

4. Planning the Physical Environment

The basic structure of the proposed way of playing is a projector, a surface to receive the projection and a camera, which is used to "see" what happens inside the projection. Different camera positions related to the projector offers different possibilities of visual interpretation. For the proposed work, the camera is positioned close and above the projector (Figure 3). Depending on the projection cone, it may be necessary to move the camera a few centimeters above or below the projector to avoid regions where the element of the body that creates the shadow (a hand for example), cover the most of the shadow. In the experiments made, with some simple adjustments it was possible to remove any important "blind spot".

Depending on game design, the size and shape of the surface may also influence the gameplay.

The surface that receives the projection can be irregular, provided that it allows the player to project his shadow on it. Irregularities on the surface may not interfere much in the game, provided that the shape of the shadow does not change much. In some cases, the presence of irregularities on the surface provides a natural variety of gameplay, where the player needs to adapt to the irregular surface. Another important point is that the projection's surface doesn't need to be completely parallel to the projection. Projections on inclined surfaces such as a table can provide an interesting alternative for some games.

The space between the projector and the projection is also relevant. A large distance between the projector and the surface that receives the projection allows a bigger game space, forcing the player to move more during a match or allowing several players in the same space. Depending on the height of the projection, the game may require the player to jump, so that his shadow reaches certain areas.

5. Shadow and Gameplay

One of the first questions that arise in relation to the interface proposed relates to why not identify the body of the player instead of his projected shadow. The reason is simple; is easier for the player to follow the game through its projected shadow. The player looks at the shadow that he generate, which is in the play field . If the camera is used to detect the player it would be necessary a much larger space to play. Other problems and limitations concerning image detection are:

- Brightness of the environment: Low light can make the identification of color and shape harder. Since projector generates the shadow, environment light is not a big problem.
- Player size: A child near the projector can generate a shadow similar of the shadow of a adult. This can balance the game, giving the child a chance against an adult.
- Size of the projection: Depending on the surface that receives the projection, the projection may be much larger than the player or much smaller (the projection in a table, for a board game). Many games can be adapted to be played in both large and small projections. In these games, instead of using the body, the player can use objects such a pencil or pen. This flexibility is not easily achieved using other detection methods.

The shadow detection can give the following information for gameplay:

- X and Y coordinates of the projected shadow: the center of the bounding box around the shadow can be used to calculate X and Y.
- Z coordinate: the shadow increase and decrease in size can indicate a movement in Z.
- Shadow area: can be calculated by defining the minimal bounding box around the shadow.
- Shape of the shadow: simple shapes can be used to send commands for the gae
- Speed of the movement of the shadow: can be easily calculated for X, Y and Z coordinates.

6. A Game Example

Five game design proposals were developed during the study. One of the designs is a game called Shadow Volleyball.

In the Shadow Volleyball each player stands on one side of the projection, outside the projection field but close enough so your hands and forearms can reach the projection region. The projection should be taller than

2 meters to allow jumps. The match starts with one player moving his hand as in a normal volleyball game. A virtual ball is created and move towards the other player considering the strength and position of the player's initial movement. The opponent hit the ball with his shadow. The players needs only to move their arms, and may also jump or crouch to catch the ball.

The game calculates the size of the shadow in order to prevent the player to increase its shadow using the body to cover a larger area of the screen. Figure shows a gameplay of the game.

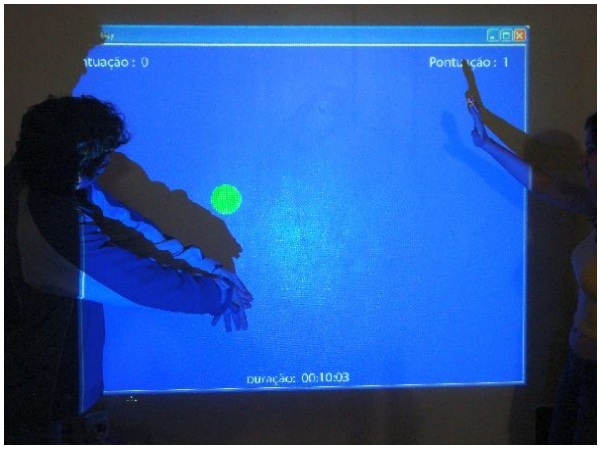


Figure 3: Shadow Volleyball.

7. Conclusions and Future Works

Due to the low cost of "Pocket Projectors" and despite the relatively low quality of the projected images, the appeals of low energy consumption and portability can make pocket projectors extremely popular in a short time. Coupled with the fact that mobile phones with projectors are already a reality, and the fact that most notebooks already have a build-in camera, it's possible to say that the hardware platform for games using shadow detection will be common very soon.

Another definitely important issue is the game experience that can be provided by the proposed interface. Initial experiments shows that the ability to interact with large projections excites the majority of players. Another motivating factor is the need to move the body to play. Unlike the Wii controller, which really does not need broad movements, the proposed interface can force players to move more their bodies.

As further work, the interface can be improved with the identification of the form of the shadow, allowing the creation of board games, where the projection of the shadow of physical pieces on the surface allow the computer to identify each piece and implement features of movement and functionality. Another field of study is the identification of gestures based on the shadow. Due to the flattening of the form done by the shadow, not all hand gestures can be adequately captured, however, due to the great definition achieved by the projection of the shadow, which hardly shows blurred regions, the detection of gestures can be more precise.

Acknowledgements

The authors would like to thank FAPERJ.

References

ROJAS, P., 2004. World's smallest color video projector. *Engadget* [online]. Available from: www.engadget.com/2009/06/11/researchers-ditch-dlp-develop-oled-panel-based-mini-projector/ [Accessed 20 June 2009]

MILLER, R., 2009. Researchers ditch DLP, develop OLED panel-based mini projector. *Engadget* [online]. Available from: www.engadget.com/2009/06/11/researchers-ditch-dlp-develop-oled-panel-based-mini-projector/ [Accessed 20 June 2009]

CAPTAIN, S., 2008. Microvision preps first laser pocket projector. *Popsci.com* [online]. Available from: www.engadget.com/2008/06/13/worlds-first-projector-cellphone-is-also-an-iphone-clone-in-ro/ [Accessed 20 June 2009]

RICKER, T., 2008. World's first projector cellphone is also an iPhone clone, in Rome. *Engadget* [online]. Available from: www.popsci.com/article/2008-01/microvision-preps-first-laser-pocket-projector [Accessed 20 June 2009]

MICROSOFT, 2009. Project Natal. *Xbox.com* [online]. Available from: www.xbox.com/en-US/live/projectnatal/ [Accessed 18 july 2009]

KUMAR, S. AND SEGEN, J., 2003. Gesture-based input interface system with shadow detection. United States Patent 6624833. Publication Date : 09/23/2003.

SHOEMAKER, G., TANG, A., BOOTH K. S., 2007. Shadow Reaching : A new Perspective on Interaction for large wall displays. ACM 978-1-59593-679-2/07/0010.

APPERLEY, M., MCLEOD, L., MASOODIAN, M., PAINE, L., PHILLIPS, M., ROGERS, B., and THOMSON, K. 2003. Use of video shadow for small group interaction awareness on a large interactive display surface. In *Proceedings of the Fourth Australasian User interface Conference on User interfaces 2003 - Volume 18* (Adelaide, Australia). R. Biddle and B. Thomas, Eds. ACM International Conference Proceeding Series, vol. 36. Australian Computer Society, Darlinghurst, Australia, 81-90.

CONCI, A., AZEVEDO, E. Computação Gráfica – Teoria e Prática, Volume 1. Rio de Janeiro : Campus, 2003.

BILOW, C., 2007. Interface orientation using shadows, MICROSOFT CORPORATION (Redmond, WA, US), 20070300182.

SERRA, J., 1983. Image Analysis and Mathematical Morphology. [S.I.] : Academic Press.

A Trivial Study Case of the Application of Ontologies in Electronic Games

Alex F. V. Machado[1] Fernando Naufel do Amaral[2] Esteban W. Clua[1]

[1] MediaLab / UFF (Universidade Federal Fluminense)
[2] LLaRC (Laboratório de Lógica e Representação do Conhecimento), PURO (Pólo Universitário de Rio das Ostras) e UFF (Universidade Federal Fluminense)

Figure 1: *Ontological Trail*: a game developed through the integration of ontologies and XNA.

Abstract

This paper proposes the creation of a simple architecture to separate programming tasks from resource allocation tasks. It demonstrates the integration of a knowledge base modeled in DL (Description Logic) with the XNA Game Engine, using the OwlDotNetApi library. The main goal is to demonstrate a trivial use case of ontologies in the domain of games. We have implemented a 2D adventure game using this technique, which we then compare with classical methods of development.

Keywords: ontologies, description logic, XNA

Authors' contact:
alexcataguases@hotmail.com
fnaufel@ic.uff.br
esteban@ic.uff.br

1. Introduction

Techniques derived from ontological knowledge bases have proven efficiency for storing information for application in several domains. In the development of electronic games, we may cite dialog creation, textual story generation and AI for adaptive game planning.

This article presents a different application of ontologies: positioning NPCs in an environment and assigning their behavior. We intend to separate the developer's work into programming tasks and level design tasks. During programming, the developer defines procedures related to file loading (graphics, sound, etc.), device control (video, keyboard input, etc.), main game components (collision detection, AI modules, etc.). During level design, the developer defines the NPCs to be inserted in the environment, their habitats (i.e., their positions) and their behavior.

2. Related Work

The main application of ontologies in game design is story flow planning. In [SANCHEZ-RUIZ 2007] the authors present an adaptive approach to game AI through case-based planning and ontological knowledge extracted from the game's environment. There, it is shown that ontology-based extraction results in strategies whose utilization is easier than those returned by classical mechanisms using only case-based planning.

Like first order logic (using mainly PROLOG), ontologies can also be used for dialog creation in games. In [PEINADO, GERVÁS and DÍAZ-AGUDO 2004] the combination of expedients such as ontologies and formal inference (as featured in Description Logics) has ensured the generation of semantically correct texts. In fact, this mechanism has been explored in automatic story generation influenced by the player's actions. The design of the ontology has allowed measuring the semantic distance between narrative functions for the generation of meaningful stories.

Although current work on games and ontologies demonstrates that this integration can be fruitful, usually little detail is presented about it. Likewise, little information is given about the specification of the knowledge bases and the technologies involved. The present work intends to fill this gap.

3. Ontologies, OWL and DL

Designing an ontology amounts to representing human knowledge in expressive fashion while keeping computational complexity at a minimum.

One of the main ontology languages is the Web Ontology Language (OWL) [W3C 2009]. OWL offers a way to build a vocabulary for the specification of a problem domain, including a set of constructs to define restrictions. Among the main features of OWL are included: Classes ("things" in the domain of interest); Relations (relationships that may occur between things); Properties (attributes that things may have).

There are different species of OWL (such as OWL Full, OWL Lite and OWL DL), which vary in complexity and expressivity. The approach presented here is based on OWL DL (Description Logic), which corresponds to a decidable fragment of first order logic [BAADER et. al., 2007]. This species of OWL allows for the processing of complex ontologies with acceptable computational overhead.

4. Tools

The game developed here has been based solely on free software tools. The development environment has been Visual Studio, the base language has been C#, the graphical library has been XNA. Protégé has been used for designing the ontology, and the OwlDotNetApi library has been used to integrate some of those technologies.

We have selected the .NET platform as the main development tool because it supports RAD (Rapid Application Development) and because it allows for interoperability between multiple languages [LIBERTY 2001].

XNA Game Studio Express is an API in the .NET platform that allows easy access to peripherals (such as the keyboard), to the graphics hardware, to audio control and data storage (in files or in databases). This API can also be used as a basis for game development for the XBox 360 console.

For ontology design we have chosen Protégé, which is currently the most popular and powerful tool for this purpose [MULLER 2008]. Protégé can be used to specify domain models and knowledge bases. It allows the creation, manipulation and visualization of knowledge in several different formats, including OWL.

The OwlDotNetApi library [MULLER 2008] is a tool written in C# that offers support to .NET applications. It has the following characteristics:

- It is a C#-based interpreter for the .NET platform;
- It is compatible with the OWL specifications;
- It can be used with any .NET language;
- It can produce directed graphs.

The OwlDotNetApi is available as a precompiled freeware DLL library.

5. Game Description

In order to develop a simple instance of integration of DL and XNA exploring the advantages of ontologies, we have implemented a strategy game. This mechanics is characterized by its emphasis on analysis and reflection in the search for the most appropriate tactics. These are games in which there is an evident component related to territorial or material conquest [SATO and CARDOSO 2008]. We have selected this style because of the diversity in the possible actions by the NPCs (NonPlayer Characters).

The game is called *Ontological Trail* (Figure 1). In it, *Onto*, the main character, must find the three magical relics scattered on *Naufel Island* so he can revive his beloved, who has recently passed away on the island. But he must dodge the wild animals living there (which may or may not attack him). In short, it is a simple game where the player must control a character and avoid or confront certain NPCs while exploring a virtual world.

The software we have developed is highly scalable. Some noticeable characteristics of the NPCs include the following:

- each NPC has a *type*, an attribute which may be defined in the system. This type may be, for example, *rabbit*, *tiger*, *buzzard*, or *dolphin.*
- each NPC has a *habitat*, an attribute which may be defined in the system. This habitat may be, for example, *sea*, *coast, jungle, land* or *world* (meaning any habitat).
- each type of NPC has a random *number of instances,* between 3 and 10, scattered throughout its habitat.
- each NPC has a random *position* inside its habitat.
- each type of NPC has an *alignment*, an attribute which may be defined in the system. The alignment may be one of the following: *hostile* (the NPC attacks the main character), *coward* (the NPC runs from the main character), or neutral (the NPC emits sounds upon meeting the main character, but does not attack or run away).

These characteristics are important, as they figure in the links between XNA and the ontology, as described in the next sections.

6. Defining the Ontology

Defining an ontology is not an easy problem [MULLER 2008]. The main steps in the specification of the ontology for the game *Ontological Trail* were:

- establishing the scope and the main goals of the ontology;

- defining the conventions for naming classes and properties;
- enumerating the main concepts and classes;

These principles have been applied to *Ontological Trail* as follows: the scope of the ontology include the animals in the jungle, their habitats and their behavior (alignment). All the classes were named after nouns, and all the properties had names that were prefixed with the verb form *has* (as in *hasAlignment*). The main classes were organized as shown in Figure 2.

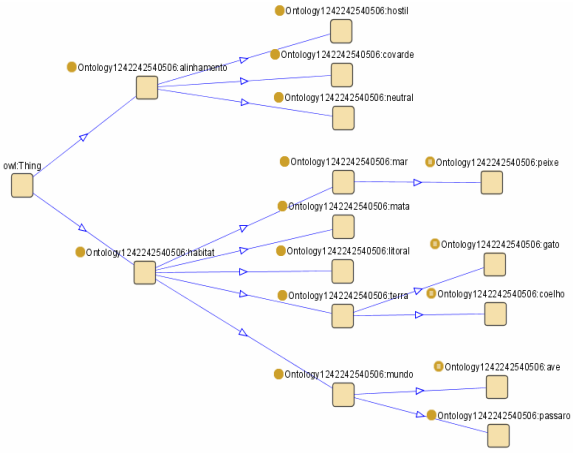


Figure 2: Ontology used in *Ontological Trail*

7. Integration

The following steps have been taken during the implementation of the software: preliminary study of Description Logic; preliminary study of C# and XNA; integration of OwlDotNetApi and C#; definition of the game mechanics; development of the ontology; development of the game using XNA; final integration.

One of the main steps (for its importance and its complexity) was the integration of C# with the OWL knowledge base. This is explained by the need to use methods *ChildNode( )* and *ParentNode( )* (Figure 4) to extract the relationships defined in DL.

8. Implementation and Tests

Ontological Trail was developed in Visual Studio and compiled for the PC (Figure 3).

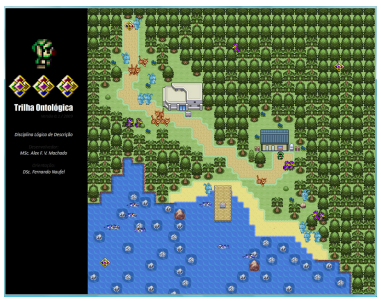


Figure 3: *Ontological Trail* interface

The main classes and components in the developed software were (Figure 4):

- *animal*: contains the attributes and methods related to the behavior (such as collision detection for the maze walls and for the main character), to the alignment modules (behavior associated to each kind of alignment) and to the rendering of the NPCs. It also handles the external files containing sprites (sequences of images composing a character) and sounds.
- *habitat.owl*: knowledge base containing the associations necessary for creating an NPC.
- *owlDotNetApi:* responsible for integrating C# and Owl. Among its main classes we note:
  - *OwlParser:* interprets an OWL knowledge base;
  - *OwlGraph*: generates a connected graph from the knowledge base;
  - *OwlEdgeCollection*: represents a collection of edges. This class maps the identification of each edge into an object of type *OwlEdge* in a list;
  - *OwlEdge:* represents an edge connecting two nodes. It is needed to verify relationships;
  - *OwlNode*: represents a node in the graph.
- *habitat*: delimits each type of habitat (sea, land, etc.) and enables the generation of the set of NPCs in the game. Uses the *OwlDotNetApi* class to communicate with the *Habitat.owl* component.
- *game:* main class in the game. Integrates and instantiates all classes in the system.

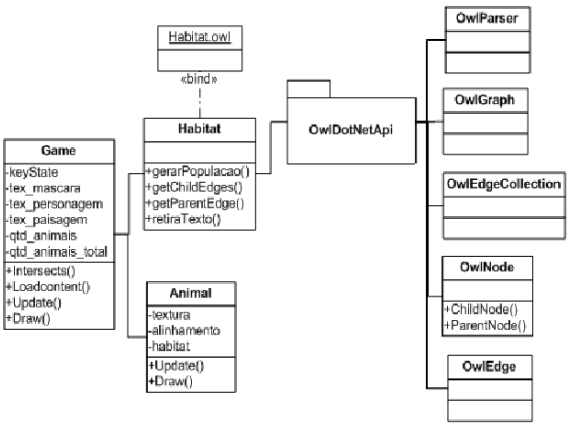


Figure 4: Class diagram

As a consequence of this integration, the developer is free to conduct a substantial part of the level design using Protégé. As shown in Figure 5, it is possible to visualize the NPCs and their habitats in hierarchical form, modify their alignment (changing the restrictions associated to their respective classes) and alter their habitats (by dragging and dropping). To insert a new

NPC type, all that is needed is to include its sprite in the project and create a new class in Protégé.

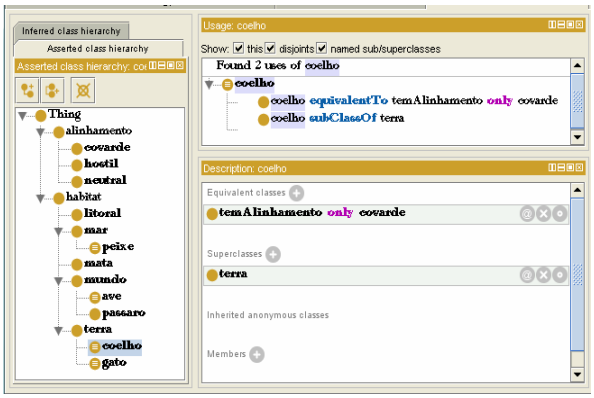


Figure 5: Class hierarchy and equivalences for class *Rabbit* in the game ontology visualized in Protégé

9. Comparison with Classical Models

For this work, we define *classical models* (in the domain of strategy games) as the technique of creating characters in dynamic fashion (using vectors of classes for predefined characters), using hardcoded information (without loading external constants for support).

The difference between this classical model and the ontology-based model is the use of a knowledge base for handling resources (in this case, NPCs).

From these considerations, it is possible to identify the following advantages of our proposed, ontology-based model over the classical models:

- easier visualization of data and relationships (due to the hierarchical representation);
- advantages inherited from OOP, such as encapsulation, abstraction and inheritance;
- easier level design (as the developer is relieved from programming tasks and may focus instead on resource management tasks);
- easier, more agile maintenance;
- an environment suited to the production of new information related to the resources, as OWL DL reasoners (e.g., Pellet [CLARK & PARSIA 2009]) may be used to infer relationships that were only implicit in the ontology, allowing for semi-automatic refinement of the knowledge base.

10. Conclusion

Although the advantages of the application of ontologies to the domain of games are being widely explored (as in [SANCHEZ-RUIZ 2007] and [PEINADO, GERVÁS and DÍAZ-AGUDO 2004]), little has been detailed about a trivial integration, let alone using free software tools.

The present work demonstrates a step-by-step implementation, from the definition of an ontological knowledge base to the implementation of the main classes of a strategy game. It aims, therefore, at serving as a basic reference for developers interested in starting this integration.

Acknowledgments

This work is partially supported by FAPERJ (Fundação de Amparo à Pesquisa do Estado do Rio de Janeiro) in E-26/112.038/2008 process and CEFET-MG (Centro Federal de Educação Tecnológica de Minas Gerais).

References

BAADER, F., CALVANESE, D., MCGUINNESS, D., NARDI, D., PATEL-SCHNEIDER, P., EDS., 2007. The Description Logic Handbook. 2nd edition. Cambridge University Press.

CLARKE & PARSIA, 2009. http://clarkparsia.com/pellet, last visited July 22, 2009.

LIBERTY, J. Programming C#. O'Reilly, 2001.

MULLER, R., 2008. *Ontologies in Automation.* Vienna University of Technology. Master Thesis.

SANCHEZ-RUIZ A. ET AL, 2007. *Game AI for a Turn-based Strategy Game with Plan Adaptation and Ontology-based retrieval*. Association for the Advancement of Artificial Intelligence.

SATO, A. K. O., CARDOSO, M. V., 2008. *Além do gênero: uma possibilidade para a classificação de jogos*. Proceedings of SBGames'08.

PEINADO, F., GERVÁS, P., DÍAZ-AGUDO, B., 2004. *A Description Logic Ontology for Fairy Tale Generation*. Proceedings of the Workshop on Language Resources for Linguistic Creativity, LREC'04.

W3C, 2009. http://www.w3.org/TR/2009/CR-owl2-conformance-20090611, last visited July 22, 2009.

Alunos como *designers*: relato de experiência para aprendizagem de Linguagens Formais e Autômatos

Ricardo L. Binsfeld[1] Rodrigo Watanabe[1] Rômulo C. Silva[1] Izaura M. Carelli[1]

[1]Universidade Estadual do Oeste do Paraná, DETAE, Brasil

Resumo: Prensky (2008) defende que os melhores designers de jogos educacionais são os próprios estudantes. O objetivo deste artigo é relatar a experiência de design de um jogo educacional pelos acadêmicos que já cursaram a disciplina de Linguagens Formais e Autômatos (LFA), na sua segunda versão, incorporando as funcionalidades levantadas no teste de usabilidade da sua primeira versão.

Palavras-chaves: jogos educacionais; aprendizagem de LFA; aluno como game designer

Autores:
{ricardobins, rodwatanabe, romulocesarsilva, imcarelli}@gmail.com

1. Introdução

Atualmente, a disciplina de Linguagens Formais e Autômatos (LFA), um dos pilares na formação de um bacharel em ciência da computação, cumpre dupla função: (1) apoiar outros aspectos teóricos da Ciência da Computação, como por exemplo, decidibilidade, computabilidade e complexidade computacional e (2) fundamentar diversas aplicações computacionais, tais como processamento de linguagens, reconhecimento de padrões e modelagem de sistemas (Menezes, 2001).

LFA descreve a parte da Teoria da Computação correspondente ao uso de máquinas abstratas para o estudo dos problemas que podem ser resolvidos por sistemas computacionais automáticos. Tem como objetivo compreender linguagens geradas por gramáticas, escrever gramáticas que representem linguagens e compreender o potencial e o limite de diferentes tipos de máquinas teóricas para manipulação de linguagens.

Porém não é tarefa fácil levar alunos a entenderem e se apropriarem dos fundamentos dessa área de conhecimento. Para auxiliá-los na assimilação dos conceitos relacionados a LFA é recomendável o uso de atividades que possibilitem a manipulação de premissas que compõem esses conceitos.

O uso de recursos computacionais para a criação de objetos de aprendizagem (OA) é amplamente adotada pelos professores em qualquer nível educacional. Um OA é considerado uma unidade de instrução/ensino que é reutilizável (Wiley, 2000), isto inclui todo o tipo de artefato digital, incluindo jogos educacionais.

Se integrados a prática pedagógica, os jogos propiciam um ambiente de aprendizagem cativante e eficaz, devido ao seu caráter lúdico.

Prensky (2008) defende que os melhores desenvolvedores de jogos educacionais são os próprios estudantes, que aprenderam determinado conteúdo e conhecem as dificuldades na aprendizagem, podendo, desse modo, identificar as melhores soluções.

O objetivo deste artigo é relatar a experiência no desenvolvimento de um jogo educacional, AutomataDefense, demonstrando que o design pode ser elaborado pelos próprios acadêmicos que já tenham se apropriado da temática a ser abordada pelo jogo criando ferramentas de aprendizagem para outros acadêmicos.

Na seção 2, discute-se como jogos educacionais podem ser utilizados para aprendizagem de LFA. Na seção 3, relata-se todo o processo do design do AutomataDefense 1.0, o teste de usabilidade e a implementação da sua continuidade.

2. Jogos Educacionais para a aprendizagem de LFA

Como LFA demanda a aprendizagem de conceitos complexos, não é simples desenvolver qualquer tipo de OA. Um levantamento de OAs identificou somente simuladores de autômatos e visualização da execução de algoritmos relacionados a autômatos através de computação gráfica, por exemplo, a ferramenta JFLAP (JFLAP, 2009). Não se referencia jogos educacionais relacionados à disciplina de LFA.

Como os jogos educacionais podem auxiliar os acadêmicos de Ciência da Computação a vivenciar a complexidade da disciplina LFA numa linguagem que lhe é conhecida, a linguagem de jogos, associado à proposta de Prensky (2008), surgiu a ideia de desenvolver um jogo com esta finalidade, pois os jogos podem ser uma nova metodologia de aprendizagem, que permite mudar o paradigma educacional centrado no professor para centrado no aluno, possibilitando que eles internalizem conceitos fundamentais desta disciplina.

3. Alunos como *Designers*

Seguindo a teoria de Prensky, o acadêmico Watanabe (2008) desenvolveu o jogo AutomataDefense, o qual auxilia no aprendizado de conceitos relacionados à disciplina de Linguagens Formais e Autômatos, mais especificamente o conceito de autômatos finitos determinísticos, permitindo a assimilação de regras, a compreensão do contexto envolvendo este conceito, e a prática do mesmo de modo interativo e atrativo. Este jogo foi implementado em Adobe Flex, que fornece as estruturas necessárias para a criação de uma interface dinâmica e atrativa.

O jogo AutomataDefense é classificado como do tipo *Tower Defense*. De modo geral, os jogos deste tipo são considerados como jogos de estratégia, porque os resultados obtidos no jogo dependem diretamente do planejamento estratégico aplicado pelo jogador. De um ponto de vista mais amplo, pode-se classificá-lo também como um mini-game, isto é, jogo de curta duração, que para fins educacionais, possui uma vantagem, pois permite abordar conceitos mais específicos de uma determinada área.

Tendo como objetivo auxiliar os alunos na aprendizagem de um tópico de LFA, mais especificamente, autômatos finitos determinísticos, implementou-se um editor de autômatos em que o jogador cria autômatos e posteriormente associa-os às torres que passam a atacar somente as criaturas que são reconhecidas pelo referido autômato.

3.1 AutomataDefense 1.0

Os jogos desse estilo são, geralmente, compostos basicamente por um tabuleiro, que possui uma entrada e uma saída (ver Figura 1) diversas criaturas, as quais entram no tabuleiro no decorrer do jogo e buscam, de algum modo, alcançar a saída do mesmo; e torres têm a capacidade de atacar as criaturas e podem ser colocadas aleatoriamente no tabuleiro.

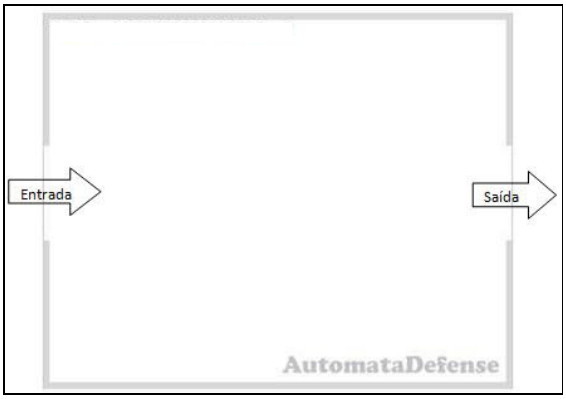


Figura 1 – Layout do tabuleiro do AutomatoDefense

O objetivo do jogo é impedir que algumas destas criaturas alcancem a saída do tabuleiro, para isso, o jogador deve distribuir as torres que atacam estas criaturas pelo tabuleiro, geralmente formando labirintos, para dificultar a movimentação das mesmas e destruí-las a tempo. Uma regra fundamental é que não se pode bloquear completamente o acesso das criaturas à saída do tabuleiro, para não perder a idéia do jogo.

Há diferentes tipos de torres disponíveis no jogo, como mostrado na Figura 2 que possuem características distintas, isto é, vantagens e desvantagens, em que, por exemplo, algumas podem ter uma melhor média de tiros por segundo, outras maior área de danos, outras que diminuem a velocidade das criaturas dentre outras.

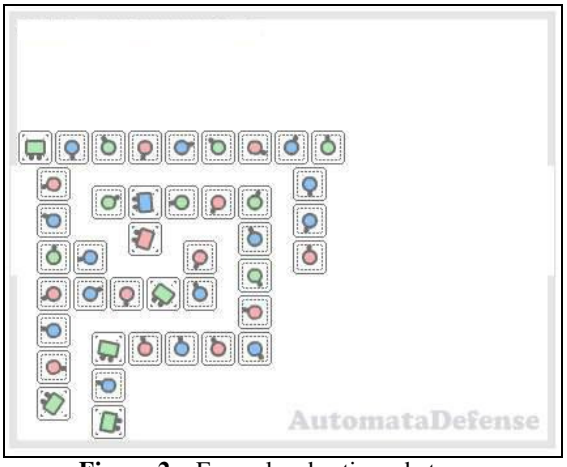


Figura 2 – Exemplos dos tipos de torres

O jogo é bastante dinâmico envolvendo alto grau de estratégia e administração dos recursos disponíveis, tanto que não há uma única solução e nem mesmo uma melhor solução para completar o objetivo do jogo.

O jogo AutomataDefense 1.0, além de possuir funcionalidades particulares para atender o objetivo de relacioná-lo com o conceito de autômatos finitos determinísticos, herda todas as características pertencentes à categoria Tower Defense.

Cada uma das criaturas que tenta atravessar o tabuleiro possui uma palavra relacionada. Há duas categorias de criaturas: monstros (Figura 3) e civis (Figura 4).

Uma das principais estratégias do jogo é a criação de autômatos finitos que reconheçam apenas palavras associadas aos monstros, pois civis não devem ser atacados. Os autômatos então criados são associados a torres que serão posicionadas no tabuleiro.

Figura 3 – Exemplos de monstros

Figura 4 - Exemplos de civis

Assim, outra funcionalidade do jogo é o editor de autômatos finitos determinísticos, mostrado na Figura 5 a partir do qual o jogador pode criar e editar autômatos a qualquer momento do jogo. Cada torre adicionada ao tabuleiro é associada a um único autômato, a qual passa a atacar somente as criaturas cuja palavra seja reconhecida pelo autômato associado. Cada civil atacado ou monstro que alcance a saída do tabuleiro são descontados pontos de vida.

O editor de autômatos não permite a inclusão de não-determinismo nos autômatos, impedindo que o jogador adicione uma transição de estado lendo um símbolo já utilizado em outra transição a partir do mesmo estado.

Figura 5 – Editor de autômatos finitos determinísticos

No AutomataDefense, o jogador é desafiado a criar autômatos finitos determinísticos que reconheçam apenas as palavras associadas a monstros. Essa habilidade está diretamente relacionada à capacidade de projetar autômatos para reconhecer linguagens específicas, frequentemente exigida de alunos da disciplina LFA.

O AutomataDefense, como os demais jogos, possui algumas regras fundamentais para controlar as ações do jogador, com a finalidade de torná-lo competitivo e desafiador.

Após o início do jogo, as invasões dos monstros no tabuleiro ocorrem em intervalos constantes de tempo, sendo que, a cada invasão, a dificuldade do jogo aumenta, portanto os monstros ficam mais difíceis de serem destruídos. O *level* do jogo começa em zero, aumentando em uma unidade a cada invasão dos monstros.

Inicialmente, o jogador possui uma determinada quantia de *gold*, sendo que esse valor aumenta a cada monstro destruído, variando conforme o *level* do jogo. Há ações no jogo que exigem o uso desse dinheiro virtual por possuírem um custo associado a elas, como, por exemplo, a construção de torres ou, até mesmo, inserção de novos elementos em um autômato. Esse valor deve ser administrado da melhor maneira possível, pois ele faz parte da estratégia do jogo e afetando diretamente o seu desempenho do mesmo.

Além do *gold*, o jogador também possui uma pontuação, chamada *score*, que varia conforme suas ações no jogo. Essa pontuação não afeta diretamente o jogo, ela tem apenas a finalidade de permitir que o jogador avalie suas estratégias.

Conforme a idéia do jogo, deve-se evitar que os monstros cheguem à saída do tabuleiro e, do mesmo modo, evitar que os civis sejam atacados pelas torres. Para que se mantenha este objetivo, descontam-se pontos de vida do jogador para cada monstro que escapa ou civil que seja abatido. Inicialmente o jogador possui um valor fixo de vidas, definido como vinte, porém pode ser alterado. Caso esse valor chegue a zero, o jogador perde o jogo, isto é, implica em *Game Over*, sendo essa a única maneira de ocorrer derrota.

O fim do jogo, com vitória, ocorre quando o jogador atingir o *level* máximo, definido atualmente como trinta, ou, com derrota, quando ocorre *Game Over*, como descrito anteriormente.

3.2 Teste de usabilidade no AutomataDefense 1.0

Foi realizado teste de usabilidade com acadêmicos que estão cursando a disciplina de LFA, cujos resultados foram publicados em (Silva et al, 2009).

O teste foi realizado com cinco participantes, sendo dividido em três etapas:

- Aplicação de questionário para obtenção do perfil do jogador
- Entrega de documento com as regras do jogo e tempo de 1 hora para testar o jogo
- Aplicação de questionário de avaliação do jogo

O teste revelou que a maioria dos participantes gostou do jogo e o avaliou como divertido (Silva et al, 2009).

Embora todos participantes tenham afirmado que as regras do jogo estavam claras, quatro dos cinco participantes tiveram dúvidas de como jogar, apenas no início, devido à falta de familiaridade com o editor de autômatos. A interface foi avaliada como boa pela maioria. Os acadêmicos sugeriram as seguintes melhorias na interface:

- alteração da cor das transições dos estados (atualmente cinza) para uma cor mais visível;

adicionar opção *limpar autômato*, apagando todo o autômato já desenhado; e
- apresentar opção para salvar os autômatos desenhados em memória secundária para não perdê-los ao fim do jogo, podendo recuperá-los em um jogo futuro.

Quanto ao conteúdo do jogo, os acadêmicos sugeriram:
- A criação de fases no jogo em que seja necessário o projeto de autômatos mais complexos;
- A criação de níveis diferentes de monstros;
- A possibilidade de aumentar o nível de ataque das torres.

3.3 AutomataDefense 2.0

Em consonância com a premissa de Prensky de que o aluno é o melhor *designer* de jogo educacional, outro acadêmico aprimorará e ampliará o AutomataDefense 1.0, incorporando as funcionalidades identificadas no teste de usabilidade anteriormente descritas, incluindo duas novas fases implementadas, seguindo a mesma metodologia (descrita na seção 3.1).

As novas fases abordarão dois outros conceitos de linguagens formais: autômatos finitos não-determinísticos e autômatos com pilha. Essa alteração amplia a abrangência de conceitos relacionados a LFA, buscando facilitar a aprendizagem dos alunos.

O grau de complexidade das novas fases demandará a modificação do editor de autômatos, permitindo não somente a criação de autômatos finitos determinísticos, mas também de autômatos finitos não-determinísticos e autômatos com pilha. Além disso, serão feitas adequações nos métodos de geração das palavras às criaturas para que essas sejam representativas quanto ao reconhecimento pelos novos tipos de autômatos.

3. Conclusão

Os resultados obtidos no teste de usabilidade da primeira versão do jogo AutomataDefense comprovam a proposta de Prensky (2008), em que ele defende os acadêmicos como designers de jogos educacionais. A partir desses resultados, optou-se por dar continuidade, incluindo na nova versão as funcionalidades sugeridas e a inclusão de novas fases, abordando outros conceitos de LFA.

Agradecimentos

Os autores agradecem ao ITAI e PTI pelo apoio ao desenvolvimento deste projeto.

Referências

JFLAP, 2009. JFLAP 6.4. DISPONÍVEL EM: HTTP://WWW.JFLAP.ORG. ACESSO EM: 27 DE JULHO DE 2009.

MENEZES, P. B., 2001. LINGUAGENS FORMAIS E AUTÔMATOS. SAGRA LUZZATTO, PORTO ALEGRE, 4 EDITION.

PRENSKY, M., 2008. STUDENTS AS DESIGNERS AND CREATORS OF EDUCATIONAL COMPUTER GAMES: WHO ELSE? BRITISH JOURNAL OF EDUCATIONAL TECHNOLOGY.

SILVA, R. C.; WATANABE, R.; CARELLI, I. M., 2009. AUTOMATADEFENSE: JOGO EDUCACIONAL PARA APOIO EM LINGUAGENS FORMAIS E AUTÔMATOS. IN: SCGAMES 2009 FLORIANÓPOLIS.

VIEIRA, N. J., 2006. INTRODUÇÃO AOS FUNDAMENTOS DA COMPUTAÇÃO. PIONEIRA THOMSON LEARNING, SÃO PAULO, 1 EDITION.

WATANABE, R., 2008. AUTOMATADEFENSE: UM JOGO EDUCATIVO PARA APOIO EM LINGUAGENS FORMAIS E AUTÔMATOS. MONOGRAFIA, UNIVERSIDADE ESTADUAL DO OESTE DO PARANÁ, FOZ DO IGUAÇU.

WILEY, D. A., 2000. CONNECTING LEARNING OBJECTS TO INSTRUCTIONAL DESIGN THEORY: A DEFINITION, A METAPHOR, AND A TAXONOMY. DISPONÍVEL EM: HTTP://REUSABILITY.ORG/READ/. ACESSO EM: 27 DE JULHO DE 2009.

An evaluation of JavaFX as a 2D game creation tool

Hamilton Lima Jr
Media Lab – UFF

Fábio Corato de Andrade
Media Lab - UFF

Anselmo Montenegro
Media Lab - UFF

Esteban Clua
Media Lab - UFF

Abstract

With the current growth in the user experience and the existence of multiple publishing platforms, the investigation of new game creation tools that simplify the development process, is important to reduce costs and increase the overall quality of the products.

Based on this perspective, we present an analysis of the JavaFX technology as a tool for 2D game development. For instance, we will focus the evaluation on the following features: deployment, scripting support, vector graphics support, flexible main loop, sprite caching, collision handling, audio support and distribution license.

Keywords: 2d games, JavaFX, tool evaluation, RIA

Authors' contact:
{hlima,anselmo,esteban}@ic.uff.br
fabio.corato@ig.com.br

1. Introduction

JavaFX [JAVAFX] is a GUI (Graphic User Interface) framework created by Sun Microsystems, based on a script language that merges XML definitions with embedded JavaScript-like code. JavaFX can use pure Java classes integrated in the scripts, which allows the enhancement of existing Java applications with a modern look and feel. With the perspective of a rich user experience, our study is based on the creation of a video game.

This analysis is based on the experience of creating a side scrolling platform video game using the JavaFX script technology. During the experiment of the game creation several decisions were made in order to accommodate the technology and the expected results of the game, most of these decisions are described in the paper separated in two main topics: The process of development and issues found during this process.

At the process of development topic, we will go over the integration with the design team, the sprites caching management, the organization of the scripts files and the collision handling. The Issues found topic will describe the current audio support of JavaFX, the difficulties to deploy the game and some license restrictions.

2. Development process

For this experience we created a port of an existing game submitted to the Global Game Jam 2009, called Tiny Soldiers the Rise of Mosquito [TINYSOLDIERS], that is a side scrolling game created in XNA.

2.1 Game main loop

Most of the available samples of JavaFX are organized in only one big script due to the simplicity of the samples, in our case we will enforce that the separation of script files will allow an object oriented organization of the script files. Our main script will be the Main.fx file that will have all the declaration of the instances used in the game.

First of all this script will have the instance of the Game object that is an extension of the Stage class.

The Game.fx class uses a TimeLine [JAVAFXAPI] object to control the game main loop; by using this type of object we can control the speed of the game by changing the time parameter of the Keyframe object inside the TimeLine

```
public class Game extends Stage {
 public var tick: Timeline = Timeline {
   repeatCount: Timeline.INDEFINITE
   keyFrames: [ KeyFrame {
       time: 10ms
       action: function() {
         mainLoop();
   }}]
 };

 public function mainLoop(){ }
 public function play(){ tick.play(); }
}
```

Code 1: *Creation of the main loop*

The "KeyFrame" object have an attribute "time" that is from the type Duration, that allows the usage of milliseconds, seconds and minutes or combination of the three. Language features like this add simplicity to the code and make the development process more intuitive.

The mainLoop() method check for objects that extends the "Updatable" class and call update() method of each one propagating the game tick for the objects that need update in the game.

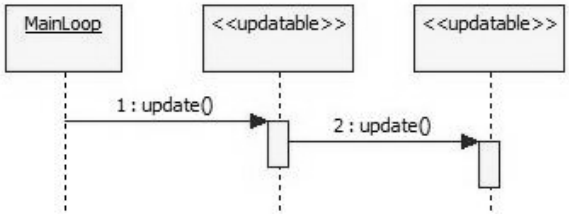


Figure 1: *Main loop sequence diagram*

In order to use the "Game" class, the Main.fx script creates a "Game" instance and call the play() method, this starts the timeline and keep the game in constant loop.

```
var game: Game = Game {
  title: "Tiny Soldiers ... "
  x: 0
  y: 50
  width: 800
  height: 600
  scene: Scene {
    content: bind currentGroup
  }
  fullScreen: false
}

function run(__ARGS__ : String[]) {
  game.play();
  soundtrack.play();
}
```

Code 2: *The Instance of the Game class*

The run() method of the "Game" instance is the JavaFX implementation of the Java main() method. In order to create the initial state of the "Game" object, all attribute changes and instance attribute creation should be defined inside the curly brackets.

2.1 Design team integration

Game development teams need someone to fill the artistic role. This role will create the environment, the GUI, the characters and NPC's (non player characters) and all visual behavior of the game. To support this role JavaFX offers support to uncompressed and compressed bitmaps files. In addition to images support, a production suite integrates JavaFX with design tools as Adobe Photoshop [PHOTOSHOP] and Adobe Illustrator [ILLUSTRATOR], beside that also offer a converter that reads SVG [SVG] files and convert then to FXZ files, that are the standard format used for images definition at JavaFX and can be read directly in the code.

As the design team can work in parallel with the programming team, we can have colored rectangles working as game characters during the development process; what can be achieved within most of existing game frameworks. With the use an SVG files converted to FXZ format is possible to add a unique identifier attribute to individual elements of drawings, usually know as ID, and these are the reference used in the script to manipulate the images that can be changed by the design team without any sort of change in the code.

The Figure 2 shows the character that was created in the SVG format and imported to the JavaFX project. After importing the SVG to the project we can use the method lookup() to retrieve references of individual parts of the image, and manipulate it, see a sample in the Code 3.

```
(player1.lookup("JFX:body")
   as Rectangle).fill = Color.BLUE;
```

Code 3: *Changing one object fill color*

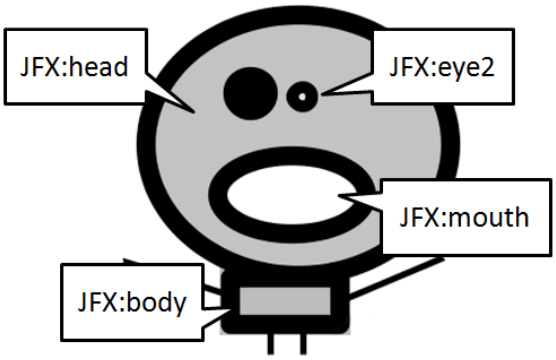


Figure 2: *Imported SVG file*

This approach allow the use of the FXZ files as integration artifact that won't need any intervention from the programming team in order to work properly in the game, this integration allow the use of complex objects and have its visual aspect changed by programming instead of having multiple sprites for the different states of the visual object.

2.2 Sprite caching

Vector images in the SVG format, are compressed with ZIP algorithm to create the FXZ files. SVG files can be as simple as a circle with a center point and a radius, or can be as complex as hundreds of shapes and points, build together to make complex illustrations, that will be read and interpreted by JavaFX every time the FXZ file is used by the game.

When complex illustrations composed by several shapes and fills, are used in the JavaFX scripts, a call to FXDLoader.load() will force the parsing of the FXZ file, as most of the times some illustrations are reused in the game, there is a need to avoid all this parsing every time the illustration is used, The Code 4 show the caching mechanism implemented to load the FXZ files only once and enable the reuse of the objects.

```
public class NodeFromFXZPool {
 var cache: HashMap = new HashMap();
 public function get(source:String):Node{
  var content:
    Node = cache.get(source) as Node;

  if( content == null){
    content =
      FXDLoader.load(source) as Node;
    cache.put(source, content);
  }
  return Duplicator.duplicate(content);
 }
}
```

Code 4: *Sprite caching implementation*

The key of the caching mechanism implementation is the Duplicator [JAVAFXAPI] class that uses and existing Node definition and create an independent copy of it.

2.3 Scripts Organization

In order to organize the code in classes we separated the classes in .fx files, but this created an problem when defining the different Scene [JAVAFXAPI] objects of the game, because we need to change the main game instance in order to indicate Scene changes. This would be simple to achieve if the scope of the event handling methods belong to the objects created, but in JavaFX the scope of the created methods belong to the script where the method were created.

In this example we have the definition of an instance of the HowToPlayGroup that will be show as current scene of the game and when the onClick method is call the currentGroup is changed to a value that is declared in the Main.fx

```
var howToPlayScene: HowToPlayGroup =
HowToPlayGroup {
    onClick: function(){
        currentGroup = menuScene;
    }
}
```

Code 5: *Use setting the Group callback*

In order to create this we build a callback solution in the scene definitions to avoid coupling with the main script,

```
public class HowToPlayGroup extends Group
{
  public var onClick: function(): Void;
  ...
  onMouseClicked: function(e){
    if( onClick != null ){
      onClick();
    }
  }
  ...
}
```

Code 6: *Creating the callback in the Group*

With this callback strategy scenes can be treated as independent artifacts that can be created and tested without the main script, reducing external dependencies and the coupling to the main Stage.

2.4 Collision Handling

In JavaFX games in order to check the collision of game elements we use the Rectangle.intersect() method that is provided with the language. There is a possibility of having more complex collision handling, this need additional implementation, that could be iterating over the existing points from an imported SVG file or from a script based polygon.

One workaround to this restriction is presented by Silveira Neto [NETO, SILVEIRA 2008], and we can see at Figure 3, where a bounding box is created at the coordinates x=4 and y=25 inside the game object itself, so part of the object really overlaps the collision target when the collision happens, offering to the player the visual feedback of the collision.

Figure 3: *Bounding box smaller than the image*

3. Issues

3.1 Audio support

The current version of JavaFX use a video/audio decoder created by On2 [ON2]. This decoder offers support to multiple video formats, and claims that offers support to MP3 files. After some tests with different MP3 files encoded with different frequencies and different quality, JavaFX wasn't able to play most of the combinations; Table 1 shows some tested combinations.

Frequency	Encoding	Duration	File size	Result
48000Hz	128bits	> 1sec	456K	Failure
48000Hz	**96bits**	**> 1 sec**	**341K**	**Success**
48000Hz	32bits	> 1sec	114K	Failure
48000Hz	16bits	> 1 sec	57K	Failure

Table 1: *MP3 combinations tests*

In order to solve this MP3 restriction, we implemented a new version of MediaPlayer class, integrating the Jlayer library [JLAYER] objects in AbstractAsyncOperation objects. With this solution all the combinations described in the Table 1 was play with success. The same solution can be used to add support to OGG, WAV or any other audio format.

3.2 Deploy of the game

We tested the Applet and Web Start [WEBSTART] deploy of JavaFX applications. For the Applet or the Web Start deploy the user needs to download the JavaFX Runtime environment that can't be released with the application due to license restrictions, that force the end user to be online at the first run of the game. This restriction is a roadblock to the usage of JavaFX as solution to create standalone games, where the users don't need internet connection to play.

Using the Web Start solution the end user is forced to handle dialogs in English, without an option to translate to the user's language; this is a serious restriction for publishing games to the general public in special for the Brazilian market. Other issue on using the Web Start solution is the fact that even with Java Runtime Environment installed Web Start application files are not automatically executed, what adds an extra complexity to the end users, in order to execute the Web Start file.

Another issue when using the JavaFX as an Applet is the fact that the whole applet must be downloaded before the user can start the interaction, this generate a high level of frustration due to the fact that Applet download don't give the user a feedback of the percentage of the download. A Java Game engine named Pulpcore [PULPCORE] based on applets solved this issue, for plain Java Applet games, by creating a small applet with less than 200k in size, which loads the real application applet, and can show to the user a feedback of the percentage of the game download.

4. Related work

Despite the fact that JavaFX is new and still with some bugs, there some casual games developed, there is a Pacman clone [PACMAN] and the Brick Breaker [BRICK] that can be found at the JavaFX documentation.

5. Conclusions

Based on the fact that JavaFX offers a full integration with existing Java code, we can assume that JavaFX has a good chance to be the next natural GUI framework choice for Java games and applications.

Related to the evaluation itself, we can conclude that JavaFX can be used to create game applications with some restrictions. The current audio support has restrictions related to MP3 files and no OGG files support. Only controlled environments where the users can make sure that access to the Internet is available, is the expected deployment scenario in order to download the JavaFX runtime environment.

Follow the summary of the JavaFX evaluation using the 2D game development challenges table:

Deployment	Online required, >10mb runtime download
Scripting support	Full with JavaFX script
Vector graphics support	Can be imported to framework script
Flexible main loop	Created with TimeLine
Sprite caching	Can be done
Collision handling	Rectangles only
Audio support	Poor mp3 support, no OGG
Distribution license	Can't distribute standalone runtime, need download

Table 2: Summary of JavaFX evaluation

We see as expansions to this research the investigation of multi-player games created with web technologies especially with JavaFX. Other future research related to JavaFX would be the investigation of 3D possibilities, in particular the integration with Jmonkey or Java3D engine.

References

BRICK, Brick Breaker JavaFX game sample, http://javafx.com/samples/BrickBreaker

ILLUSTRATOR, http://www.adobe.com/products/illustrator

JAVAFX, http://javafx.com

JAVAFXAPI, JavaFX API documentation, http://java.sun.com/javafx/1/docs/api/index.html

JLAYER, Pure Java mp3 library, http://www.javazoom.net/javalayer/javalayer.html

NETO, SILVEIRA 2008, How to create a RPG like game, http://silveiraneto.net/2008/12/08/javafx-how-to-create-a-rpg-like-game

ON2, On2 technologies, http://www.on2.com

PACMAN, JavaFX Pacman clone, http://www.javafxgame.com/javafx-pac-man-article-5

PHOTOSHOP, http://www.adobe.com/products/photoshop

PULPCORE, http://www.interactivepulp.com/pulpcore/

SVG, http://www.w3.org/TR/SVG

TINYSOLDIERS, Tiny Soldiers the Rise of Mosquito, Global Game Jam 2009, http://globalgamejam.org/games/tiny-soldiers-rise-mosquitos

WEBSTART, http://java.sun.com/javase/technologies/desktop/javawebstart/index.jsp

Aplicação de Redes Neurais e Algoritmos Genéticos em Jogos de Corrida

André Grahl Pereira Cesar Tadeu Pozzer Erick Baptista Passos*
Marcos Cordeiro d'Ornellas

UFSM, Departamento de Eletrônica e Computação, Brazil
*UFF, Instituto de Computação, Brazil

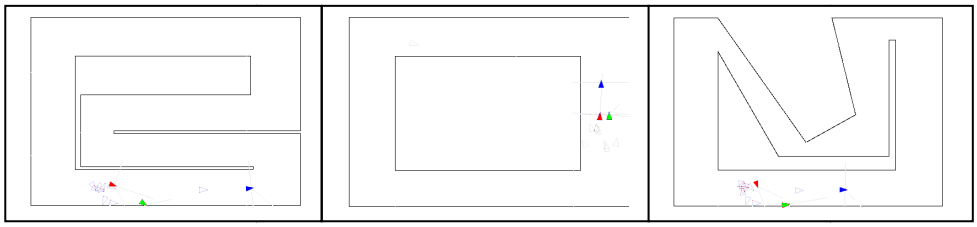


Figura 1: Variação de pistas, modelo de população e veículos utilizados nas simulações.

Abstract

Este artigo apresenta um modelo alternativo de inteligência artificial para jogos de corrida. O problema abordado é a criação de um agente para controle de um automóvel de corrida com comportamento semelhante ao de um ser humano, suscetível a erros e adaptável a situações emergentes no decorrer do jogo. O sistema proposto combina redes neurais artificiais, treinamento *off-line* por meio de algoritmos genéticos e adaptação em *real-time*.

Keywords: redes neurais, algoritmos genéticos, jogos de corrida, aprendizagem em tempo real

Authors' contact:
{grahl, pozzer, ornellas}@inf.ufsm.br
@epassos@ic.uff.br

1. Introdução

Jogos eletrônicos são uma indústria em expansão. A busca por melhorias é constante, seja no contexto de realismo físico e gráfico, como no caso de jogos como Crysis [Crysiseu 2007], ou na elaboração de histórias com grande imersão, como no caso de Máfia 2, que possui um roteiro de 700 páginas [Ekberg 2009].

Nessa busca por realismo nos jogos, o aperfeiçoamento gráfico e físico, bem como a complexidade das histórias e do game-design são abordagens importantes, mas outro foco de especial interesse atual é o desenvolvimento de inteligência artificial (IA) mais convincente.

Neste contexto, o uso de técnicas clássicas de inteligência artificial, como máquina de estados finitos, apesar de eficientes, demonstram pouco realismo [Buckland 2002]. O objetivo é desenvolver agentes mais realistas, que não sejam apenas eficientes e acertem sempre, e sim cometam erros, preferencialmente de maneira semelhante a um ser humano. Técnicas adaptativas de IA são uma alternativa neste contexto. Redes neurais e algoritmos genéticos apresentam características desejáveis como: adaptação, aprendizagem e diversidade de reações em situações idênticas propiciada pela variação dos pesos da rede.

No caso especifico de jogos de corrida, o desenvolvimento de NPC's (*Non-player characters*) é uma situação ainda mais complexa. Para que o NPC gerado seja um bom adversário, uma série de qualidades são necessárias tais como: velocidade de reação, previsão, criatividade, aprendizagem e adaptação [Togelius 2005].

A abordagem mais utilizada neste tipo de problema é uma combinação de linhas de corrida com máquina de estados finitos [Biasillo 2002]. A linha de corrida é utilizada como guia para o veículo, delimitando o caminho pelo qual ele deve percorrer a pista. Neste caso o NPC já conhece previamente a pista antes mesmo da corrida ser iniciada. Pode-se desenvolver uma linha de corrida perfeita criando um NPC invencível, o que foge dos objetivos de uma inteligência artificial realista.

Este cenário se mostra ideal para utilização de técnicas adaptativas de IA com aprendizagem em tempo real. No entanto, autores como Spronck [Spronck 2003] não as considera como candidatas para esta utilização, pois não satisfazem quatro dos requisitos apresentados pelo autor: rapidez, eficácia, robustez e eficiência. No entanto, outros autores tais como Yannakakis [Yannakakis 2005], que desenvolveu um jogo utilizando o modelo predador-presa utilizando um algoritmo evolutivo no aprendizado on-line de agentes controlados pelo computador através de uma rede neural, e Crocomo [Crocomo 2006] que elaborou um jogo em que as estratégias adotadas pelos agentes controlados pelo computador se adaptam de acordo com as estratégias utilizadas pelo jogador a cada vez

que este joga, demonstraram que é viável a utilização de aprendizagem *on-line* em jogos eletrônicos.

Neste trabalho é apresentado um modelo alternativo de inteligência artificial para jogos de corrida. O sistema tem como fundamentos a criação de agentes com comportamento semelhante ao de um ser humano, suscetível a erros e adaptável, para o controle de um carro de corrida em um jogo.

Para tanto, o sistema usa uma combinação de treinamento *off-line* e adaptação *on-line*. São usadas informações simuladas que são semelhantes às disponíveis para um jogador ou piloto, ou seja, não são necessários *waypoints*, como na maioria das abordagens anteriores para esse problema. O agente utiliza como entrada de dados apenas as informações obtidas por um campo de visão baseado em traçamento de raios.

Na seção 2 são apresentados trabalhos relacionados e diferenças em relação ao atual trabalho. Na seção 3 é apresentada uma breve descrição protótipo de jogo de corrida utilizado nos experimentos. Na seção 4 é discutido o sistema de aprendizagem e de adaptação. As seções 5 e 6 discutem aspectos relevantes referentes a rede neural e ao algoritmo genético. Na seção 7 são apresentados dados e comentários sobre os experimentos realizados. A seção 8 contém a conclusão e trabalhos futuros.

2. Trabalhos Relacionados

Em Togelius [Togelius 2005] também foi utilizada uma arquitetura de redes neurais multicamadas e algoritmos genéticos para a criação de um agente piloto de automóveis. Foram utilizadas três configurações de pistas diferentes com um veículo sem adversário e dois tipos de sensores: os de pistas e dos *waypoints* que recebiam distância e direções dos objetivos. O objetivo desse trabalho foi experimentar a capacidade de especificação da rede neural e dos sensores utilizados. Entretanto, dado o tipo de entrada de dados utilizada, que incluiu o uso de *waypoints*, observou-se que os parâmetros ganhavam configurações muito específicas para cada pista, sendo impraticável usar um agente treinado para uma pista em outra. O nosso sistema utiliza informações equivalentes à inferência visual de um piloto, sendo possível treinar o agente com uma pista e utilizá-lo em qualquer outra.

Stanley [Stanley 2005] desenvolveu um sistema que utiliza redes neurais e algoritmos genéticos como previsor de colisões. Os testes foram conduzidos em uma pista com o simulador RARS [RARS 2009], onde o sistema recebe como entradas a distância de objetos e das laterais da própria pista. Esses dados são inseridos na rede neural e se obtêm como saída um nível de alerta para uma possível colisão. O nosso sistema usa uma abordagem semelhante para a entrada de dados, mas faz uso de um agente piloto completo para ser utilizado em uma corrida em tempo real.

Em Wloch [Wloch 2004], utilizou-se algoritmos genéticos para a otimização de parâmetros no simulador *Formula One Challenge '99-'02*. Foi demonstrado que este é um método eficiente na melhora no desempenho e conseqüentemente na obtenção de melhores tempos.

No trabalho de Vieira [Vieira 2006], foi apresentado um sistema que utiliza pontos guia para direcionar a trajetória do veículo em uma pista seccionada. Os pontos guia são obtidos através de um sistema de aprendizagem que utiliza algoritmos genéticos na busca por melhores referências.

3. Protótipo do Jogo de Corrida

Para os experimentos, foi desenvolvido em OpenGL um simulador 2D de jogos de corrida. A pista é delimitada por barreiras sólidas, que são também identificadas pelo sistema de visão, o qual é simulado pelo traçamento de raios. Um sistema simples de simulação física foi implementado, incluindo atrito e forças.

Os veículos são representados por formas geométricas simples e três pistas de configurações distintas foram criadas para possibilitar diversos experimentos, como mostrado na Figura 1.

4. Sistema de Aprendizagem

O sistema de aprendizagem (Figura 2) é utilizado em dois momentos distintos: na adaptação *on-line* e na aprendizagem *off-line*. Em ambos os momentos, sempre existe uma população de veículos participando da simulação. Dois grupos de veículos são utilizados: os corredores e os fantasmas.

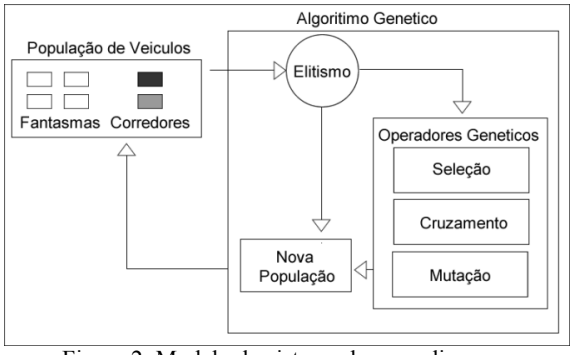


Figura 2: Modelo do sistema de aprendizagem

Os corredores são formados pelos indivíduos da população que obtiveram o melhor desempenho na geração anterior. Como as colisões entre esses são consideradas, um corredor afeta o comportamento dos demais. Esses são os indivíduos com os quais o jogador interage.

Os fantasmas são formados pelo restante da população. Entre esses não existe colisão, mas os mesmos levam em consideração os veículos do jogador e do grupo dos corredores para o seu próprio aprendizado e simulação.

4.1 Aprendizagem Off-line

Na aprendizagem *off-line*, inicialmente a população de veículos é criada aleatoriamente. Esta fase é dividida em dois momentos, antes e após a pista ser completamente percorrida.

Antes da pista ser percorrida, o valor da função de avaliação para o algoritmo genético é dado pela quantidade de *checkpoints* ultrapassados, o numero de *checkpoints* varia de pista para pista sendo usados em pontos específicos como curvas ou em fim de obstáculos, pontos que teoricamente determinariam um aprendizado. Observa-se que os *checkpoints* não são usados pelo sistema *runtime* de direção, sendo apenas condição necessária para a escolha inicial de indivíduos que "progridem" na pista, do contrário a geração inicial aleatória poderia não conter indivíduos capazes de completar uma volta, dificultando a avaliação do algoritmo genético.

Quando um indivíduo é capaz de dar uma volta completa, sua avaliação é dada pelo tempo de duração de uma volta. Após um limite de tempo, ou após todos os indivíduos completarem uma volta na pista, o algoritmo genético é então ativado para evoluir a população.

Os pesos dos neurônios de cada indivíduo são o seu genoma, sendo que a aptidão de cada um é o desempenho obtido pelo método descrito acima. Os genomas com maior aptidão (grupo elite) são diretamente transferidos para a nova população. Com essa técnica de elitismo, esses corredores farão parte da próxima geração sem sofrer influência de operandos de permutação e mutação, garantindo assim um desempenho mínimo igual ao da presente geração.

O restante da população é utilizado normalmente com o uso da permutação e mutação, incluindo o grupo elite. Após algum tempo de treinamento, são obtidos corredores com habilidades mínimas que podem ser utilizados com o mecanismo de adaptação *on-line*.

4.2 Adaptação On-line

Na adaptação *on-line*, os corredores obtidos na aprendizagem *off-line* são utilizados para um maior refinamento. Isso garante um desempenho mínimo ao sistema perante o jogador, o que é uma característica importante para jogos casuais, onde o jogador normalmente não é um *expert*. Dessa forma, também são excluídos comportamentos aleatórios ou demasiadamente inesperados, mas garantindo a variabilidade necessária para que haja candidatos de soluções suficientes para a adaptação em relação ao jogador.

5. Rede Neural

A rede neural é responsável por todo o comportamento do veículo, a cada instante recebe os dados do ambiente, processa de acordo com seu genoma e reponde com um conjunto de saídas. Graças à variedade genética dos genomas, cada veículo responde de forma diferenciada a eventos idênticos. Sua configuração é de oito entradas (uma para cada sensor), uma camada oculta com 25 neurônios e quatro saídas. As quatro saídas são os comando do veiculo virar a direita, esquerda, acelerar e frear. O sensoriamento da pista é realizado por 8 sensores de tamanho fixo presentes em cada veículo, como ilustrado na Figura 3.

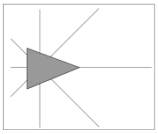


Figura 3: Configuração dos Sensores dos veículos

Quando o sensor não está tocando em nenhum objeto assume o valor de entrada -1. Quando o sensor toca em algum objeto o valor de entrada é a distancia, que é normalizada entre 1 e 0, valendo 0 quando o ponto externo do sensor toca o objeto e 1 quando o ponto interno do sensor toca o objeto.

6. Algoritmo Genético, Seleção, Cruzamentos e Mutação

O operador usado na seleção é o torneio, onde inicialmente um membro da população é escolhido aleatoriamente. Ele é então comparado com *'n'* outros membros escolhidos aleatoriamente. Como a comparação é feita com dois membros o que possuir melhor desempenho permanece e o outro é descartado. Ao final das *'n'* comparações o membro vencedor é escolhido para o cruzamento.

O cruzamento é feito através do operador de multipontos por partes, onde múltiplos pontos definidos aleatoriamente dos genomas escolhidos são trocados dentro de faixas validadas, ou seja, só podem ser trocados pesos de um mesmo neurônio (Figura 4).

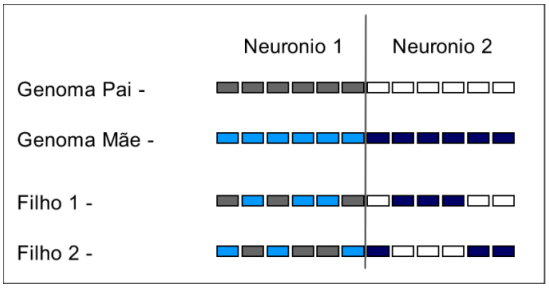


Figura 4: Cruzamento entre dois membros selecionados pelo operador mutação

O operador mutação é usado para multiplicar uma faixa aleatória da rede neural por valores aleatórios, gerando assim uma perturbação na rede neural, o que possibilita o surgimento de novas características (Figura 5).

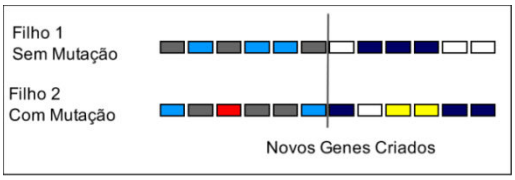


Figura 5: Mutação dos filhos gerados pelo operador cruzamento

7. Experimentos Realizados

Durante os experimentos *off-line* o sistema foi capaz de em pouco mais de nove gerações treinar indivíduos capazes de completar a pista de forma satisfatória, e em torno das 150 gerações o sistema alcançou a estabilidade (Figura 6).

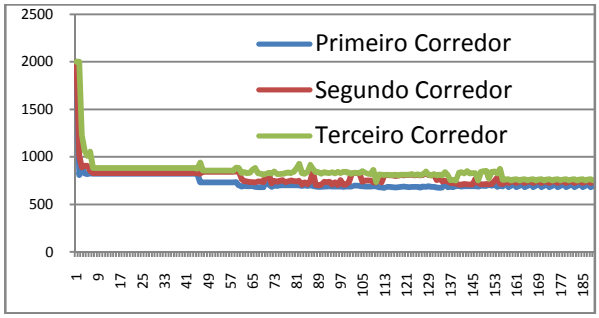


Figura 6: Desempenho dos corredores num experimento

Os corredores obtiveram comportamentos interessantes como o de buscar ultrapassagens por caminhos alternativos e o de impedir ultrapassagens fechando a frente de seus adversários. Também buscaram o caminho ótimo ao percorrer as curvas de forma fechada e fazendo as transições entre as retas da maneira mais breve possível, visando diminuir a distância total do trajeto.

Nos testes de adaptação *on-line*, resultados igualmente interessantes foram obtidos. Os corredores conseguiram evitar dificudadades impostas pelo jogador como em casos onde o jogador ficava colidindo constatemente contra os corredores. Nesta situação os corredores se adaptaram e usaram posturas evasivas, matendo distância dos adversários e assim evitando as colisões.

8. Conclusão e Trabalhos Futuros

Após a análise dos resultados obtidos, pôde-se concluir que o sistema proposto é capaz de trazer bons resultados para o problema de desenvolver um NPC que possua características como velocidade de reação, previsão, criatividade, aprendizagem e adaptação.

Para trabalhos futuros pretende-se testar esse sistema em um simulador 3D completo como um jogo de ocorrida real. Também pretende-se agregar mais complexidade à rede neural usando NEAT, que é uma tecnica que permite a evolução da topologia da rede neural, juntamente com um sistema de treinamento mais robusto, além de elaborar um método que permita agregar conhecimento a todo sistema possiblitando o aprendizado efetivo por exemplo após completar uma pista.

9. Referências

BIASILLO, G., 2002b. Racing AI Logic. RABIN, S. AIGame Programming Wisdom. New York: Charles iver Media. p.444-454.

BUCKLAND, M.,2002. AI Techniques for Game Programming. 1 ed. Premier Press , 2002.

CROCOMO, M. K., 2006. Desenvolvimento de um Jogo Adaptativo Utilizando um Algoritmo Evolutivo. *Dissertação (Graduação) – Universidade de São Paulo – Instituto de Ciências Matemática e Computação (USP-ICMC), São Carlos, SP.*

CRYSISEU, 2007. *Crysis Graphics*. Available from: http://crysiseu.com/content/view/19/43/ [Accessed 23 June 2009].

EKBERG B., 2009. *Mafia II Impressions*. Available from: http://www.gamespot.com/xbox360/action/mafia2/news.html?sid=6211649 [Accessed 23 June 2009].

RARS, 2009. *Robot Auto Racing Simulator*. Available from: http://rars.sourceforge.net/ [Accessed 24 June 2009].

SPRONCK., 2003. Online Adaptation of Game Opponent AI in Simulation and IN Practice. *In: Proceedings of the 4th International Conference on Intelligent Games and Simulation.*

STANLEY K, O., KOHL N.,SHERONY R., MIIKKULAINEN R., 2005. Neuroevolution of an automobile crash warning system.*In: Proceedings of theGenetic and Evolutionary Computation Conference (GECCO-2005), 2005.*

TOGELIUS J., LUCAS S, M., 2006. Evolving robust and specialized car racing skills.*In: Evolutionary Computation, 2006. CEC 2006. IEEE Congress on.*

VIEIRA., 2006. Um controle autônomo de um carro de corrida. *In: Brazilian Symposium on Computer Games and Digital Enterniment.*

WLOCH K, O., BENTLEY P, J., 2004. Optimising the performance of a formula one car using a genetic algorithm.*In: Proceedings of Eighth International Conference on Parallel Problem Solving From Nature, 2004, pp. 702–711.*

YANNAKAKIS, J. N., 2005. A thesis submitted to University of Edinburgh in the subject of Artificial Inteligence for the degree of Doctor of Philosophy. *In: Computer Games: Generating Interesting Interactive Opponents by the use of Evolutionary Computation*.

Bricking: Modelagem Tridimensional de Cenários de Jogos em Camadas

Raul Joaquim C. dos Santos
rauljcs@gmail.com

Selan Rodrigues dos Santos
selan@dimap.ufrn.br

Departamento de Informática e Matemática Aplicada
Universidade Federal do Rio Grande do Norte
Campus Lagoa Nova, 59072-970, Natal/RN

(a) (b) (c)

Figura 1: *Screenshot* do jogo Pathfinder, desenvolvido com a modelagem por *bricking*. Em (a) temos a identificação dos *bricks* que compõem a cena. Em (b) temos a representação correspondente texturizada dos *bricks*. Finalmente em (c) é possível observar o efeito final, com iluminação e neblina.

Resumo

In this paper we introduce a practical and simple method for three-dimensional (3D) game level design, based on an extension of the tile modelling concept. The method, called bricking, preserves the main features of tiling, such as simplicity and low memory requirements. The main advantages of this proposal is a greater power of modelling since it is possible to define more than one layer of tiles, thus allowing complex structures to be subdivided and treated as individual units, called bricks. Differently from what happens in typical pseudo-3D games based on tile modelling, the players' movements are not restricted to a single plane — thereby allowing players to have a true 3D experience. To test our proposal, we present a first-person style game whose levels were modelled with bricking, and discuss the results from this experience.

Este artigo apresenta um método prático e simples de construir modelos de cenários de jogos para ambientes tridimensionais, baseado em uma extensão do conceito de modelagem através de *tiles*. O método proposto, denominado de *bricking*, mantem as principais características do método *tiling*, como simplicidade de manipulação e economia de memória, com o benefício de uma maior poder de representação e deslocamento não mais limitado a um único plano. Como forma de validação parcial da proposta, apresentamos um jogo modelado com a técnica *bricking* e analisamos esta experiência.

Keywords: Modelagem 3D, cenários de jogos, *tiles*, detecção de colisão.

1 Introdução

Atualmente existem várias metodologias para a criação de jogos, que dividem o processo de desenvolvimento em várias etapas. Uma das etapas da criação de um jogo é a de modelagem ou representação do cenário. Um dos objetivos da modelagem de cenários é dar suporte à estória do jogo eletrônico e prender a atenção dos usuários, desta forma promovendo um maior engajamento. Além de prover a ambientação para o desenrolar do jogo, a modelagem de cenários pode auxiliar *testes de colisão* e *determinação de visibilidade* para otimização do *pipeline* de renderização [Akenine-Möller et al. 2008, Cap. 14].

A construção de cenários pode ser uma tarefa complexa, mas existe uma diversidade de métodos que auxiliam este processo, diferindo em complexidade de implementação, leque de operações suportadas e demanda de processamento [Eberly 2006]. *Tiling* é um exemplo de técnica de representação de cenário, cujas maiores qualidades são simplicidade de representação, uso otimizado de memória e boa velocidade de renderização [LaMothe 2001]. Esta técnica, utilizada em jogos mais antigos, possibilita a criação de cenários dividindo a imagem da cena em uma grade regular — os elementos repetidos da grade são carregados apenas uma única vez e replicados nos demais locais.

Apesar de originalmente bidimensional (2D), a técnica *tiling* foi adaptada com sucesso para a tridimensionalidade (3D), como no caso do clássico título Wolfenstein 3D [id Software 2009b], dando origem ao conceito de representação 2.5D ou *pseudo-3D*. Neste caso, o mapeamento continua 2D, mas cada *tile* é representado por uma malha 3D ou poliedro, possivelmente texturizado.

Apesar de ser capaz de simular um ambiente tridimensional, o conceito de 2.5D possui uma limitação referente ao tratamento de colisão: cada *tile* do ambiente contém um único objeto, o qual é tratado de maneira atômica. Desta maneira, se um determinado *tile* é definido como "sólido" não é possível transpô-lo, mesmo que sua representação gráfica seja a de uma janela aberta ou um obstáculo baixo, como um caixa. Portanto, o problema da utilização de *tiles* para modelar cenários 3D reside no fato de que, se o *tile* for composto por várias partes, não é possível tratar cada uma delas independentemente.

Para resolver o problema acima citado, propomos uma extensão da modelagem por *tiles*, denominada *bricking*, que preservas as características originais de simplicidade de representação e uso reduzido de memória pra recursos gráficos. *Bricking* é organizado em N camadas de *tiles*, com suporte para a operação de tratamento de colisão e é capaz de oferecer uma representação verdadeiramente 3D, superando as limitações das representações 2.5D. A Figura 1 apresenta *screenshots* de um jogo desenvolvido com esta nova técnica.

2 Conceitos Relacionados

Nesta seção descrevemos em mais detalhes alguns conceitos relacionados à proposta ora apresentada, como a técnica de *tiling* e detecção de colisão.

Representação de cenário

O objetivo de *tiling* é a criação de cenários de forma simples e com economia de memória. Um cenário modelado por *tiles* é formado por um conjunto de pequenas imagens que juntas geram a imagem desejada da cena [Pa 2009]. Em linhas gerais, divide-se uma imagem correspondente ao cenário em partes menores, como pequenos azulejos; depois todas as figuras distintas que foram formadas são separadas. Em seguida, rotula-se cada figura diferente, posicionando-as de maneira a reconstruir a imagem original.

A economia de memória e processamento vem do fato de que não é mais necessário carregar de uma vez toda a imagem de fundo do cenário. Basta carregar apenas as imagens (menores) correspondentes aos *tiles* distintos entre si e replicá-los onde necessário. A Figura 2 exemplifica como acontece a divisão e a (re)utilização de *tiles*.

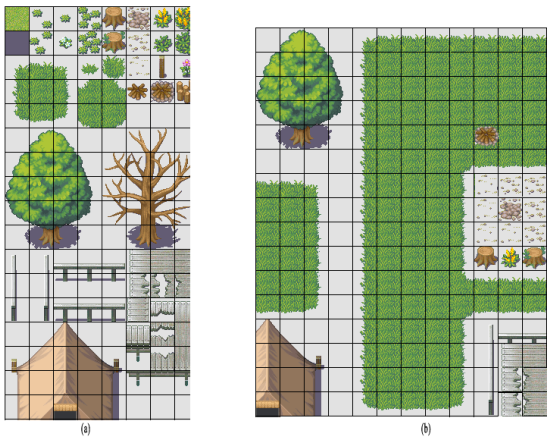


Figura 2: Em (a) temos a representação do mapa de *tiles* associado a um cenário. A imagem (b) exibe a criação de um cenário utilizando as imagens do mapa de *tiles*.

Tratamento de colisão

Com a utilização de uma grade regular para definir os *tiles*, é possível determinar a posição individual de cada *tile*. Na modelagem de um cenário de jogo é importante definir que determinados tipos de *tile* não podem ser penetrados por objetos que podem se movimentar dentro do ambiente. Portanto, a detecção da colisão ocorre quando a área ocupada por um objeto intersecta os limites da região do *tile* (ou conjunto de *tiles*) definido como 'intransitável' e que, portanto, não pode ser ocupada.

Esse mesmo mecanismo simples de detecção de colisão é usado em jogos 2.5D. A representação em 2.5D ou pseudo-3D combina uma representação 2D ou mapa de *tiles* com elementos renderizados em 3D. Exemplos de variações desta ideia são: *i)* personagens 2D atuando em frente a um cenário de fundo em 3D, como é o caso do Street Fighter IV [Capcom 2009]; *ii)* personagens poligonais 3D sobre uma plano de fundo bidimensional, como é o caso de Ragnarok Online [Gravity Co. and Myoungjin 2005]; ou *iii)* um jogo com representação completa em 3D (via *sprites* e polígonos texturizados), mas com deslocamento restrito a um único plano, como é o caso de títulos como DOOM da Id Software [id software 2009a] e Duke Nukem 3D da 3D Realms [Realms 2009].

Apesar do cenário ser renderizado em três dimensões, o controle de colisão realiza apenas a fase de descarte ou *broad phase* [Ericson 2005] entre a posição atual da câmera ou personagem e o *tile* em contato. Se o *tile* em questão for 'transitável' o deslocamento prossegue normalmente; caso contrário é indicada a colisão, independentemente da representação 3D associada ao *tile*.

A estratégia de tratamento de colisão acima descrita pode gerar situações inconsistentes com a expectativa dos jogadores e a própria realidade representada no jogo eletrônico. Por exemplo, um jogador normalmente espera ser capaz de passar por cima de um obstáculo baixo como uma caixa ou passar pela abertura de uma grande janela, ao se dirigir a tais obstáculos. Contudo, se o *tile* que contém o obstáculo fosse definido como 'intransitável' esse comportamento não seria possível, uma vez que não há tratamento de colisão por refinamento (ou *narrow phase*) dentro do *tile*. Desta maneira, o jogador não conseguiria transpor o obstáculo, mesmo que visualmente isso parecesse ser possível.

A motivação para a técnica de representação de cenário proposta neste trabalho é justamente solucionar este problema e ainda conservar as vantagens da representação de cenários por *tiling*. Pode-se afirmar que os requisitos que nortearam o design da solução proposta foram os seguintes:

- Suportar a colisão a nível de descarte;
- Utilizar algoritmos de manipulação simples;
- Prover um método com economia de memória, uma vez que os componentes seria carregados uma única vez e reutilizados quando necessários;
- Oferecer uma estrutura de dados de simples representação e manipulação; e
- Proporcionar uma maior capacidade de representação de objetos do que o método tradicional de pseudo-3D, uma vez que não estaríamos mais restritos a um único plano de navegação e representação de objetos.

3 Modelagem Bricking

O modelo de representação de cenário por *bricking* tem como objetivo estender a técnica de representação por mapa de *tiles* de maneira a suportar mais de uma plano de representação. A região de modelagem associada com o cenário é organizada em N camadas, de acordo com a necessidade de representatividade para elementos do cenário. As camadas seguem o conceito de *tiling* e são organizadas em grades regulares independentes umas das outras. Uma vez definidas, as camadas são dispostas em alturas diferentes formando uma pilha de camadas sobrepostas. Objetos podem ser inseridos em qualquer coordenada, (x,y), em uma dada camada i, definindo desta maneira um *brick*, $B_{(x,y),i}$.

A estrutura de dados básica usada para representação por *bricking* é uma matriz tridimensional. A primeira dimensão da matriz indica a quantidade de camadas; as outras duas dimensões indicam as coordenadas Cartesianas (x,y) de um *brick* dentro do plano definido por uma camada. Por exemplo, uma matriz $3 \times 4 \times 5$ terá três camadas, e cada uma delas formada por um plano que contém *bricks* distribuídos em uma matriz 4×5, conforme ilustrado na Figura 3.

Assim, em uma matriz $M[c][i][j]$ o valor c corresponde à camada do cenário, os valores de i e j correspondem às coordenadas do objeto contido na camada c, em um ambiente tridimensional. Além da matriz de representação, é necessário definir uma lista L de conjuntos de coordenadas de *bricks* que correspondem a um mesmo identificador. O objetivo desta lista é otimizar o processo de renderização, pois ao invés de percorrer a matriz de maneira sequencial trocando de contexto cada vez que um *brick* diferente fosse acessado, renderiza-se todos os *bricks* que compartilham o mesmo contexto (i.e. textura, geometria, etc.), minimizando *overhead* associado com tal troca.

A técnica proporciona a criação de um modelo de cenário com os objetos da cena identificados por *bricks*. Tendo isto em vista, a modelagem do cenário deve ser feita determinando as dimensões de um *brick* de forma a conter o maior

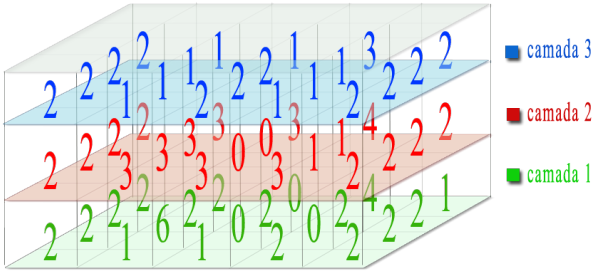


Figura 3: Representação esquemática de uma matriz correspondente a um cenário com 3 camadas, organizadas em uma grade de 4×5 *bricks*. Os números que aparecem dentro de cada *brick* são seus identificadores, ou seja, *brick* com o mesmo identificador significam objetos iguais.

objeto a ser representado no modelo, para que nenhum outro objeto do modelo ultrapasse as dimensões do *brick*. A operação de detecção de colisão da técnica é feita apenas em *broad phase*. Desta forma, o teste de colisão será feito diretamente com os *bricks* identificados como 'intransitáveis' na cena. A detecção de colisão em *narrow phase*, feita diretamente com o objeto contido no *brick*, ainda não é suportado pela técnica.

Economia de memória

Em relação ao custo de memória, a técnica mantém as características de *tiles*: os *bricks* que possuem a mesma identificação representam os mesmos objetos renderizados nas suas coordenadas correspondentes. Como os objetos são os mesmos, as texturas que forem aplicadas nestes objetos também serão as mesmas e, desta forma, só necessitarão ser carregadas uma única vez e aplicadas em todos os objetos de mesma identificação.

A modelagem de cenários por *bricking* poderá ser utilizada para subdividir partes maiores de um cenário. Por exemplo, paredes de um cenário podem ser divididas em vários 'tijolos' menores. Desta maneira, basta carregar apenas as imagens correspondentes às partes contidas nos *bricks* distintos que formam a parede e replicá-los onde for necessário.

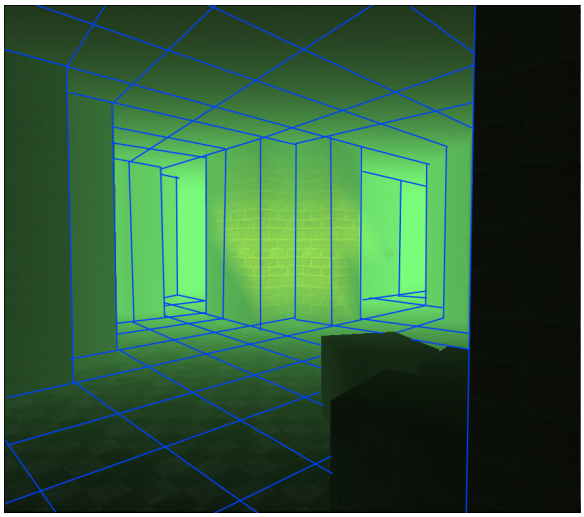
Poder de representação

Essa organização permite uma maior capacidade de representação. Por exemplo, a Figura 4-(a) apresenta uma imagem de um cenário modelado por *bricking*, porém utilizando apenas 1 camada — isso seria equivalente ao modelagem por *tiles*. Na imagem é possível notar que existe uma estrutura (em segundo plano, no centro da imagem) correspondente a uma coluna. Na Figura 4-(b), temos o mesmo cenário, desta vez organizado em 4 camadas de mesma altura. A antiga estrutura de coluna agora é representada com apenas 2 *bricks*, uma na camada superior e outro na inferior. Desta forma, se o personagem possuir uma altura de apenas 2 *bricks*, seria possível passar por "dentro" da coluna subindo no *brick* inferior. Note que esta ação não seria possível com a modelagem por *tiles*, pois a coluna ocuparia um único *tile* e seria definida como 'intransitável', mesmo que sua representação visual fosse a exibida na Figura 4-(b).

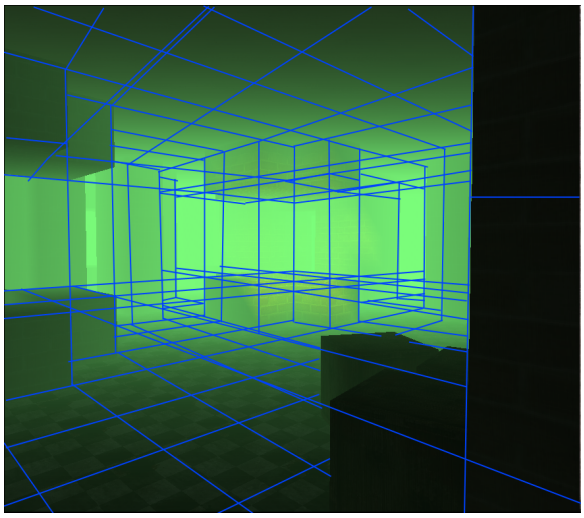
4 Resultados

Para testar o funcionamento da técnica de *bricking*, um jogo de navegação em primeira pessoa foi desenvolvido, denominado de Pathfinder [dos Santos 2009]. O jogo foi desenvolvido na linguagem C++, utilizando a biblioteca GLUT [Martz 2006].

A modelagem do cenário de jogo foi realizada com *bricking*,



(a)



(b)

Figura 4: Exemplo de um cenário modelado por *bricking*, sendo constituído de 4 camadas de mesma altura. Linhas foram sobrepostas à imagem para demonstrar a localização das camadas e suas subdivisões em *bricks*.

utilizando-se um editor de texto simples para a definição da matriz de componentes e seus recursos. A lista de conjunto de *bricks* idênticos também foi gerada manualmente. O cenário contém objetos como caixas, muros e paredes texturizados, colocados em diferentes posições no ambiente. Apresenta efeitos de neblina e de iluminação, de maneira a dar suporte ao enredo do jogo.

4.1 O Jogo

Em Pathfinding o jogador deve percorrer o cenário para coletar itens, em uma corrida contra o tempo. O cenário é relativamente complexo, contendo vários corredores, aberturas, obstáculos, colunas e entradas na forma de portas, janelas, e túneis — esta última modalidade requer que o usuário se abaixe e rasteje para atravessar.

Além das vias de acesso, o cenário possui diversas caixas, espalhadas e empilhadas, todas modeladas como um único *brick*. Especificamente, foram utilizadas 4 camadas de mesma altura para modelar o cenário.

4.2 Tratamento de colisão

O tratamento de colisão foi implementado realizando um teste para verificar se o jogador está tentando se mover para

uma região que foi determinada como 'intransitável'. Em caso afirmativo, o jogador é impedido de entrar nesta região. Caso contrário, o jogador poderá transitar livremente sobre esta área.

O protagonista possui largura inferior a um *brick* e altura inferior a 2 *bricks*. Portanto é possível passar por qualquer área livre igual ou maior do que 1×2 *bricks*, inclusive buracos nas paredes, plataformas ou janelas. A Figura 5 apresenta um *screenshot* do jogo.

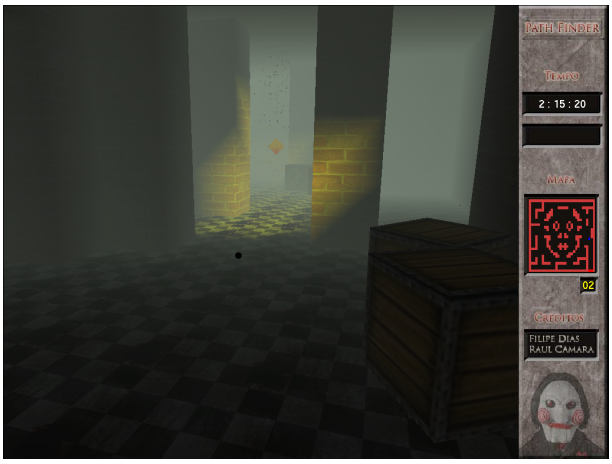


Figura 5: *Screenshot* do Pathfinder, exibindo o cenário modelado por *bricks*, e a interface com usuário (lado direito), que exibe o mapa do ambiente, a localização da chave e o tempo restante.

5 Conclusão e Trabalhos Futuros

Bricking é uma técnica de modelagem com o objetivo de prover uma forma simples de criar cenários de jogos visualmente atrativos e com baixo custo de memória. A forma de construir um modelo por *bricking* é simples, pois trata-se de uma extensão tridimensional do modelo de *tiles*, representada por uma matriz tridimensional. Além disso, jogos desenvolvidos com esta técnica serão capazes de realizar operações de detecção de colisão em alto nível (*broad phase*).

Em comparação com jogos desenvolvidos em 2.5D, especialmente os que são desenvolvidos em 2D para simular um ambiente tridimensional, nossa proposta traz benefícios em relação à detecção de colisão. O uso de *bricks* permite que a detecção de colisão seja realizada com objetos em diferentes alturas individualmente, possibilitando um comportamento mais compatível com uma tridimensionalidade real.

Contudo, a técnica ora proposta possui limitação de representação, determinada pelo tamanho dos *bricks*. Quanto maior o volume de um *brick*, menor será a precisão na detecção de colisão, caso o objeto associado ao *brick* não ocupe todo o volume.

Os próximos passos para o presente trabalho são:

- Adaptar a técnica de maneira a suportar a determinação de visibilidade. A meta é otimizar o *pipeline* de renderização, por exemplo, enviando apenas os *bricks* visíveis a partir da localização atual da câmera ou personagem.
- Definir um modelo descritivo de cena modelada por *bricking*, de maneira que seja possível desenvolver *plugins* para carregamento de cena em motores de jogos.
- Criar um editor de cenários interativo com suporte à *bricking*. Neste editor será possível definir o tamanho do modelo, escolher o número de camadas e *bricks* por camadas, definir os rótulos onde ficarão cada objeto da cena e gerar a lista representando a ordem de renderização dos objetos que compõem a cena.

Referências

Akenine-Möller, T., Haines, E., and Hoffman, N. 2008. *Real-Time Rendering*, 3rd ed. A K Peters, Ltd., Wellesley, MA, USA.

Capcom, 2009. Street Fighter IV. `http://www.capcom.com/streetfighter/`, acessado em julho 2009.

dos Santos, R. J. C., 2009. Pathfinder. `http://rauljcs.jimdo.com/arquivos-ufrn/computacao-grafica/pathfinder/`, acessado em julho 2009.

Eberly, D. H. 2006. *3D Game Engine Design: A Practical Approach to Real-Time Computer Graphics*, second ed. The Morgan Kaufmann Series in Interactive 3D Technology Game Architecture and Design. Morgan Kaufmann.

Ericson, C. 2005. *Real-Time Collision Detection*. The Morgan Kaufmann Series in Interactive 3-D Technology. Morgan Kaufmann.

Gravity Co., and Myoungjin, L., 2005. Ragnarok online. `http://www.ragnarokonline.com/`, acessado em julho 2009.

id software, 2009. DooM. `http://www.idsoftware.com`, acessado em maio 2009.

id Software, 2009. Wolfenstein 3D. `http://www.idsoftware.com/games/wolfenstein/wolf3d/`, acessado em julho 2009.

LaMothe, A. 2001. *Isometric Game Programming with DirectX 7.0*. Premier Press Game Development (Software). Muska & Lipman/Premier-Trade.

Martz, P. 2006. *OpenGL(R) Distilled*, first ed. Addison-Wesley Professional.

Pa, T., 2009. Tile based games. `http://www.tonypa.pri.ee/tbw/`, acessado em julho 2009.

Realms, D., 2009. Duke Nukem 3D. `http://www.3drealms.com`, acessado em julho 2009.

BRiGaS: Um Servidor de Jogos Pervasivo para UPnP Usando o Arcabouço BRisa

Diogo Dutra Albuquerque
IC/UFAL

Vitor Normande Vieira
IC/UFAL

João Pedro Pontes
IC/UFAL

Leandro Melo de Sales
IC/UFAL

Leandro Dias da Silva
IC/UFAL

Resumo

O protocolo UPnP (*Universal Plug and Play*) visa prover uma forma de conexão simples, robusta e interativa entre dispositivos móveis, eletro-eletrônicos e até eletrodomésticos provenientes de diversos fabricantes, provendo e facilitando a convergência de serviços e aplicações. Apesar de ser um padrão internacional emergente (2008), ele tem se tornado bastante popular na indústria, permitindo novas funcionalidades com o desenvolvimento de diversas aplicações, introduzindo a nova era de computação pervasiva. Devido a pervasividade do UPnP, que utiliza um sistema de descoberta de dispositivos, torna-se possível a integração instantânea de novos dispositivos em uma rede local. Neste trabalho, uma nova utilidade para o protocolo UPnP é apresentada: prover serviços e funcionalidades para jogos. É introduzida a especificação de um padrão para servidor de jogos pervasivo e a implementação do mesmo: o BRiGaS (BRisa Game Server). O BRiGaS estende o arcabouço BRisa para o protocolo UPnP, adicionando funcionalidades para ambientes de jogos. A pervasividade provida e a possibilidade do uso de dispositivos portáveis e móveis facilita e torna mais divertida a integração e interação entre jogadores, além de favorecer a convergência.

Keywords:: UPnP, Games, Computação Pervasiva

Author's Contact:

{diogo.comp,vitunv,joaopedro.plima,leandroal,
leandrodds}@gmail.com

1 Introdução

Devido a explosão e popularização do consumo de dispositivos móveis e a ininterrupta evolução da tecnologia móvel, novos conceitos e tecnologias surgem a cada dia. A pervasividade permite que dispositivos entrem em uma rede local automaticamente interagindo e fazendo uso de serviços providos por outros elementos do domínio da rede. O paradigma da computação pervasiva objetiva que um dispositivo, que esteja presente em qualquer ambiente, possa se comunicar com várias redes de forma transparente e interativa [Weiser 1991]. O padrão UPnP (*Universal Plug and Play*) vem sendo amplamente utilizado para flexibilizar e facilitar a integração de dispositivos em uma rede local. Pela fácil implantação e por seu sistema de descoberta de dispositivos, somados à utilização de tecnologias abertas consagradas da internet como HTTP, XML e SOAP, este padrão é uma ótima escolha para criar sistemas pervasivos. Desta forma, cada vez mais a computação móvel está promovendo a convergência de mídias, tecnologias, serviços, informações e aplicações para proporcionar mais funcionalidades e conforto aos usuários nas mais diversas áreas da sociedade.

Devido às características do UPnP discutidas anteriormente, é útil criar um servidor de jogos usando a tecnologia UPnP. Neste caso é especificado um dispositivo UPnP centrado em jogos, definindo dessa forma um servidor de jogos interoperável, aceitando clientes de diversas naturezas como celulares, PDAs[1], *Internet Tablets*, computadores *desktop*, consoles de vídeo games e qualquer outro dispositivo eletrônico capaz de implementar o padrão, como exemplificado na Figura 1. Nesse cenário, as pessoas podem jogar em rede, interagindo com o(s) servidor(es) de jogos disponível(eis) no ambiente sem a necessidade de configurar diretamente seus dispositivos.

[1] *Personal Digital Assistants*

Figura 1: *Interação com o BRisa Game Server (BRiGaS)*

Porém, o uso do padrão UPnP no universo dos jogos atualmente está apenas em *media servers* e *renderers*, controle remoto e mecanismos para facilitar o acesso às redes locais. Portanto, neste trabalho é introduzida uma nova possibilidade ao uso do padrão UPnP: prover controle e interação para jogos através de serviços providos por um dispositivo centrado em jogos. Foi então especificado um padrão para servidores de jogos UPnP e também foi feita a implementação do mesmo, de código aberto e portável que permite a interação entre diversos dispositivos, o BRiGaS (*BRisa Game Server*).

O BRiGaS está sendo desenvolvido utilizando o arcabouço BRisa [Guedes et al. 2008] para UPnP. O BRisa é um arcabouço de código livre escrito que permite a implementação de dispositivos, serviços e pontos de controle UPnP. A implementação do *Game Server* levou a questionamentos e alterações na implementação do arcabouço, influenciando e contribuindo em seu desenvolvimento. Também, levanta algumas questões sobre o funcionamento do padrão UPnP para o uso em certas aplicações, em que os dispositivos de uma rede local interajam de forma similar a jogos eletrônicos. Foi implementado um jogo de cartas como caso de teste para o servidor de jogos.

O restante deste trabalho está organizado da seguinte forma. O padrão UPnP é introduzido na Seção 2. Na Seção 3 é apresentada a especificação do BRiGas seguida pela sua implementação na Seção 4. Em seguida, teste de conceito é introduzido na Seção5. As conclusões do trabalho são discutidas na Seção 6.

2 UPnP

De acordo com Sales [Sales et al. 2008], o UPnP é uma especificação que agrega a descoberta de dispositivos com a utilização de serviços, utilizando tecnologias consagradas da internet como HTTP, XML e SOAP. A sua funcionalidade é descrita por seis passos: endereçamento, descoberta, descrição, controle, eventos e apresentação.

Considerando que o endereçamento IP dos pontos de controle e dispositivos está devidamente correto, então a descoberta dos dispositivos é realizada. Durante o processo de descoberta, a descrição do dispositivos é enviada, e o ponto de controle descobre suas informações. Os pontos de controle realizam o controle dos dispositivos através do envio de comandos. Os dispositivos se comunicam com os pontos de controle através de eventos. Finalmente após enviar e receber informações, o ponto de controle pode utilizar a apresentação para atualizar a interface com o usuário da forma desejada.

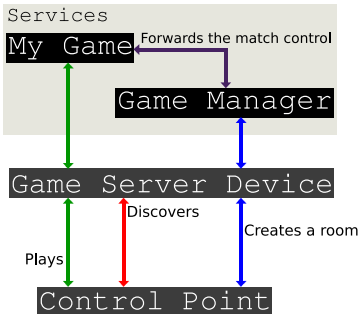


Figura 2: *Esquema de funcionamento do BRisa Game Server (BRiGaS)*

No caso do BRiGaS, um ponto de controle UPnP descobre um dispositivo *Game Server* (UPnP) na rede. Ao identificar seu tipo, o ponto de controle já pode conversar com o serviço *Game Manager*. Este último é encarregado de gerenciar os jogadores e as salas de jogos. Cada jogo neste servidor possui salas, que são criadas pelos próprios jogadores. As salas são ambientes virtuais onde os jogadores se encontram para começar uma partida. Então os pontos de controle se comunicam com o *Game Manager* para obter informações sobre salas existentes e se juntarem às mesmas ou criar novas salas. Ao começar uma partida, o *Game Manager* libera o controle da partida para o serviço do jogo em questão. Este fluxo é exemplificado na Figura 2.

2.1 BRisa

O BRisa é um arcabouço que implementa o protocolo e arquitetura UPnP, escrito na linguagem de programação Python[2]. Inicialmente, utilizou-se uma abordagem que adotou a parte de áudio/vídeo do UPnP como foco principal. Atualmente, alcançou-se um estado de arcabouço geral para o UPnP. Além disso, possui certas características como API[3] de alto nível para construir dispositivos UPnP e serviços, através da utilização de programação orientada a objeto, integração com Qt[4] [Blanchette and Summerfield 2008], Gtk[5], Glib2[6] e Ecore[7]. Neste arcabouço, também estão implementadas facilidades para *login*, configuração, *multithreading*, *networking* e possui ferramentas para trabalhar com UDP, HTTP, XML, IP e interfaces de rede.

3 Especificação do Game Server

O *Game Server* controla e possibilita a interação de dispositivos durante uma sessão, como por exemplo, criar um jogo de dominó entre quatro dispositivos. A especificação se divide em duas partes: O *Game Server Device*, que constitui o dispositivo propriamente dito, e o *Game Manager Service*, encarregado de gerenciar jogos e jogadores do servidor.

3.1 Game Server Device

No contexto de um servidor de jogos, este deve disponibilizar, ao menos, um jogo e também deve gerenciar como os jogadores podem reconhecer esses jogos. Tornando abstrato este fato para o UPnP, foi criada a seguinte temática: existe um dispositivo UPnP, denominado *Game Server*, e cada jogo dentro deste servidor é um serviço UPnP provido pelo mesmo. Isto quer dizer, que para cada jogo existente no servidor, existe um serviço relacionado, através do qual os pontos de controle se comunicarão, o que permitirá serem feitas as jogadas pelos clientes. Os serviços dos jogos não são descritos na especificação do *Game Server*, pois, cada jogo possui uma forma de comunicação, o que não permite criar uma comunicação padrão para todos os tipos de jogos. Há somente uma única exigência para os serviços de jogos: eles devem possuir os UUIDs dos jogadores participantes. Serão dados mais detalhes na próxima seção. Como este servidor deve gerenciar os jogadores e o acesso dos mesmos aos jogos, foi criado o serviço UPnP *Game Manager*, encarregado de gerenciar os jogadores conectados ao servidor, mostrar quais jogos estão disponíveis e realizar a transição dos jogadores entre os jogos e suas salas. O XML que especifica o *Game Server Device* pode ser encontrado no site [8] do projeto. Este XML possui somente o serviço *Game Manager*. Os serviços adicionais, os jogos, ficam a critério da implementação deste servidor. Um exemplo de um desses serviços de jogos é mostrado abaixo, explicado detalhadamente na Seção 5:

```
<?xml version="1.0" encoding="utf-8"?>
<root xmlns="urn:schemas-upnp-org:device-1-0">

[...]

<serviceList>
 <service>
  <serviceType>
   urn:upnp-ic-ufal-br:service:GameManager:1
  </serviceType>
  <serviceId>GameManager</serviceId>
  <SCPDURL>/GameManager/scpd.xml</SCPDURL>
  <controlURL>/GameManager/control</controlURL>
  <eventSubURL>/GameManager/eventSub</eventSubURL>
  <presentationURL>
   /GameManager/presentation
  </presentationURL>
 </service>
 <service>
  <serviceType>
   urn:ic-ufal-br:service:Truco:1
  </serviceType>
  <serviceId>Truco</serviceId>
  <SCPDURL>/Truco/scpd.xml</SCPDURL>
  <controlURL>/Truco/control</controlURL>
  <eventSubURL>/Truco/eventSub</eventSubURL>
  <presentationURL>
   /Truco/presentation
  </presentationURL>
 </service>
</serviceList>

[...]

</root>
```

Exemplo de descrição do jogo de truco

3.2 *Game Manager Service*

Como destacado anteriormente, este serviço é encarregado de gerenciar os jogos e os jogadores. Para o *Game Manager* funcionar corretamente, devem ser disponibilizados pelo *Game Server Device*, quais jogos estão disponíveis. A partir disso, este serviço criará uma estrutura, em que cada jogo possui salas, que permitirão aos jogadores interagirem. O *Game Manager Service* possui os seguintes métodos SOAP: *GetUUID*, *GetAvailableGames*, *GetAvailableRooms*, *CreateRoom*, *EnterRoom*, *LeaveRoom* e *GoodBye*. Também possui a *State Variable RoomUpdate* com a propriedade *Send Event*. O XML que o especifica pode ser encontrado no site[9] do projeto.

O método GetUUID gera um novo UUID, utilizado para identificar o jogador dentro do servidor e nos jogos. Já o método *GetAvailableGames* retorna os jogos registrados no dispositivo. É importante destacar que o nome dos jogos são separados pelo caractere ':'. Exemplo:

```
"Truco:Domino:Xadrez"
```

O método SOAP *GetAvailableRooms* recebe o nome de um jogo como parâmetro, devendo retornar as variáveis *RoomsID*, *MaxPlayers* e *NumPlayers*, como especificadas no XML. As propriedades das salas são retornadas nessas três variáveis, onde cada posição de

[2] http://www.python.org/
[3] *Application Programming Interface*
[4] http://www.qtsoftware.com/products
[5] http://www.gtk.org
[6] http://library.gnome.org/devel/glib/stable/
[7] http://wiki.enlightenment.org/index.php/Ecore
[8] http://launchpad.net/brigas/trunk/0.1/+download/example-game-server-device.xml
[9] http://launchpad.net/brigas/trunk/0.1/+download/example-game-manager-scpd.xml

cada vetor, separado pelo caractere ':' refere-se a uma sala. Exemplo:

```
RoomsID: "0:1:3"
MaxPlayers: "3:2:4"
NumPlayers: "2:1:1"
```

Neste exemplo existem três salas: a primeira sala com o identificador 0, com número máximo de jogadores 3 e com número atual de jogadores 2. O mesmo vale para as outras duas salas. Em *CreateRoom* é recebido o UUID do jogador que criou a sala, o jogo que ele pretende jogar e a quantidade máxima de jogadores na sala. O serviço *Game Manager* deve criar em sua estrutura esta sala, adicionar o UUID dado a ela e então retornar o ID da nova sala criada.

No método *EnterRoom*, o serviço deve atualizar o jogo especificado como parâmetro *Game*, adicionando o valor de UUID à sala de *RoomID*. Esse método possui uma função muito importante no *Game Manager*, pois ele é encarregado de iniciar a partida. Quando a sala chegar ao número limite de jogadores, este serviço tem de informar ao jogo que ela está cheia, e portanto que partida deve começar. Ao iniciar a partida, são passados os UUIDs de todos os jogadores para este jogo. É retornado o valor 1 em caso de sucesso. O método *LeaveRoom* deve realizar a tarefa de retirar de sua estrutura de salas o jogador com o valor de UUID, do jogo de nome *Game* e da sala com o identificador *RoomID*. Retorna o valor 1 em caso de sucesso.

O método SOAP *GoodBye* é utilizado quando um jogador se desconecta do servidor. Então é enviado seu UUID para ser retirado pelo serviço *Game Manager* da lista de jogadores. Retorna o valor 1 em caso de sucesso.

Nos métodos *CreateRoom*, *EnterRoom* e *LeaveRoom* deve ser alterado o valor da *State Variable RoomUpdate* para que os pontos de controle tenham conhecimento de uma nova atualização na lista de salas dos jogos do servidor. O novo valor da variável deve seguir o seguinte padrão:

```
"Game:RoomID:NumPlayers"
```

4 Implementação do Game Server

Para implementar o *Game Server* que foi especificado na seção anterior, utilizou-se o arcabouço BRisa. Esta implementação foi denominada de BRiGaS, que possui dois componentes principais, o *BRisa Game Server* para o lado servidor e o *BRiGaS Control Point* para o lado cliente. O arcabouço BRisa foi criado de forma que a implementação do padrão UPnP seja simples, fácil e prática.

4.1 BRisa Game Server

Para o *BRisa Game Server* ser implementado, foram utilizadas as classes *Device* e *Service* do arcabouço. Com a classe *Device* facilmente cria-se um dispositivo UPnP, como é mostrado no exemplo a seguir a criação do *Game Server Device*:

```
[...]

self.device =\
  Device(
    'urn:upnp-ic-ufal-br:device:GameServer:1',
    self._server_name,
    force_listen_url=self._listen_url,
    manufacturer='BRiGaS Team. UFAL students',
    manufacturer_url=project_page,
    model_description='An UPnP Game Server',
    model_name = 'BRisa Game Server version 0.1',
    model_number='0.1',
    model_url=project_page,
    serial_number='01'.rjust(7, '0'))

[...]
```

Implementação do dispositivo

Uma funcionalidade interessante do *BRisa Game Server* é o modo como ele reconhece os jogos disponíveis no servidor. Foi criada uma estrutura para que terceiros possam extender o *BRisa Game Server* com a implementação de seus próprios jogos, e o dispositivo seja capaz de identificar automaticamente esses jogos, como se fossem *plugins*. Para que isso seja possível, o *BRisa Game Server* possui duas classes extensíveis, das quais todo novo jogo no servidor deve fazer a extensão de pelo menos uma dessas classes. Essas classes pertencem ao módulo *games* chamadas *BaseRoom* e *BaseGame*. A classe *BaseRoom* não é obrigatória de extensão, pois ela já possui as funcionalidades básicas de uma sala de jogo, mas às vezes pode ser necessário implementar algo específico. A classe *BaseGame* deve sempre ser estendida por um novo jogo no servidor, pois ela possui métodos usados pelo *Game Manager* que devem ser implementados, que são *start_new_match* e *add_room*. É importante ressaltar também que conforme a especificação, cada jogo no servidor é um novo serviço disponível no dispositivo. Então, os jogos implementados para o *BRisa Game Server* devem estender a classe *Service* do arcabouço BRisa.

```
[...]

class MyGame(BaseGame):

  # Start new match
  def start_new_match(self, room):
    print "New Match!!!"

  # Create new room
  def add_room(self, uuid, max_players):
    print "New Room created!"

[...]
```

Extensão dos métodos da classe BaseGame

Para que esses jogos sejam reconhecidos automaticamente pelo *BRisa Game Server* eles devem obedecer algumas regras. A primeira delas é que o módulo do jogo precisa estar dentro do módulo games do *BRisa Game Server*, e este módulo deve possuir uma variável chamada *service_name*, que deve ser o nome da classe criada para o serviço do jogo. É a partir dessa variável que a classe é instanciada dinamicamente. Uma forma de implementação pode ser a seguinte:

brisa_game_server/games/mygame/__init__.py:

```
from brisa_game_server.games.mygame.impl\
import MyGame
from brisa_game_server.games.mygame.impl\
import service_name
```

Inicialização do módulo do jogo

brisa_game_server/games/mygame/impl.py:

```
[...]

service_name = 'MyGame'

class MyGame(BaseGame, Service):
  def __init__(self):
    Service.__init__(self, service_name,
                     service_type, '',
                     scpd_xml_path)
    BaseGame.__init__(self)

[...]
```

Implementação do serviço

4.2 BRiGaS Control Point

No lado cliente temos um encapsulamento da API de ponto de controle do BRisa, uma API facilitadora. O *BRiGaS Control Point* faz o acesso direto ao serviço *Game Manager*, abstraindo toda a lógica do UPnP para a aplicação cliente. A chamada dos métodos SOAP do serviço são encapsuladas, criando uma interface simples de utilização do *Game Manager*. Os eventos gerados pelo serviço *Game Manager* também são encapsulados e tratados internamente. Este também possui uma forma de comunicação com seus

utilizadores. O BRiGaS CP (*BRiGaS Control Point*) tem a possibilidade de inscrição para eventos, onde os interessados em saber das atualizações dos estados do mesmo podem ser notificados. A inscrição é feita passando-se uma função ou método que se deseja executar quando da ocorrência deste evento.

brisa_game_server/games/mygame/impl.py:

```
[...]

from brisa_game_server.control_points.game_manager \
import GameManagerCP

[...]

def fun():
 print 'Hello!'

[...]

control_point = GameManagerCP()
control_point.event_subscribe('RoomUpdate', fun)

[...]
```

Exemplo de uso da API de eventos

Quando o evento *RoomUpdate* acontecer a função *fun* será executada e será escrito *'Hello!'* na tela. Seguindo este modelo do *BRiGaS Control Point*, podem ser criados pontos de controle para qualquer jogo que o servidor possua. Desta forma, os clientes podem ser criados sem a utilização do protocolo UPnP diretamente, provendo uma abstração para que sejam criados clientes de forma fácil e ágil. Um estudo de caso é apresentado a seguir para ilustrar o uso do BRiGaS.

5 Estudo de caso: BRiGaS de Truco

A prova de conceito utilizada é um jogo de cartas originado na América do Sul chamado Truco, por conta de sua popularidade e simplicidade, o que facilita sua implementação. O Truco é jogado em duplas e somente parte das cartas de um baralho é utilizada. A principal característica do jogo é a competição por pontos, fazendo com que as duplas aumentem o valor de uma partida compelindo a dupla adversária a desistir, o que exige interação e negociação entre jogadores.

A idéia foi construir uma aplicação portátil para que os jogadores pudessem executar o Truco em qualquer sistema e arquitetura que, por sua vez, pudesse executar o arcabouço BRisa. A implementação SOA (*Service Oriented Architecture*) do serviço no *Game Server* envia notificações de acordo com a seguinte estratégia: quando uma certa variável definida no serviço é alterada, o serviço manda mensagens informando os estados específicos da mudança. A mensagem é formatada no padrão de *string SOAP* permitindo que aplicações do jogo em qualquer linguagem usem a informação, ou seja, a comunicação é definida usando definições de serviço para permitir a portabilidade da aplicação.

Nesse estudo de caso uma primeira aplicação cliente foi desenvolvida para dispositivos *Internet Tablet* da Nokia usando a plataforma *Maemo*[10] por ser um sistema Linux. A linguagem Python foi utilizada juntamente com o arcabouço BRisa e a API Edje[11] foi utilizada para desenvolver a interface com o usuário, por prover bons efeitos visuais e por fácil uso na implementação. A interface do jogo de Truco é ilustrada na Figura 3.

6 Conclusões e Trabalhos Futuros

Neste artigo é descrita a criação de uma especificação para servidores de jogos utilizando o padrão UPnP. A partir desta especificação, foi implementado o BRiGaS (*BRisa Game Server*). Com a especificação, definimos um novo dispositivo (*UPnP device*), o *Game Server*, e um serviço, o *Game Manager*. Para provar

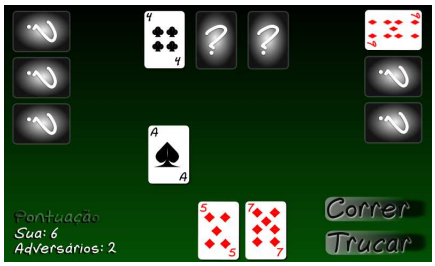


Figura 3: *Interface do jogo de Truco*

os conceitos dessa especificação, a ferramenta foi implementada utilizando o arcabouço BRisa para o padrão UPnP. Além disso, foi apresentada a implementação de um jogo de Truco como estudo de caso.

Com o desenvolvimento do trabalho, algumas necessidades e também possibilidades de aperfeiçoamento forma observados, inclusive sobre limitações do próprio padrão UPnP. Primeiramente, é necessária a criação de uma extensão para o protocolo, pois devido às limitações do mesmo com relação ao envio de eventos *multicast*, vários pacotes desnecessários são enviados pela rede. Sabe-se que é dessa forma que os pontos de controle recebem a notificação das mudanças de variáveis de estado. Ainda não há uma forma de envio de eventos *unicast*. Portanto, a nova extensão deve especificar uma forma de envio de eventos *unicast*, ou seja, fazer a comunicação entre um dispositivo e um ponto de controle específico.

Em relação ao próprio BRiGaS, há a necessidade da especificação de um esquema de segurança e criptografia no tráfego de algumas mensagem enviadas e recebidas pelo *Game Server* para dificultar a interceptação de informações dos jogos. Outro trabalho futuro é prover a identificação única dos jogadores para criar perfis no servidor, estabelecendo *ranking* e reputação. Uma solução é utilizar a extensão UPnP-UP (*UPnP User Profile*) [Sales et al. 2008] para prover essa funcionalidade.

Com a identificação do usuário através do UPnP-UP pode-se também criar um sistema de personalização do jogo, onde o jogador pode mudar suas configurações no servidor e esta personalização pode ser carregada para qualquer servidor de jogos que o jogador se conectar. Ainda com o UPnP-UP, pode ser feito um esquema de recomendação de jogos, oponentes e parceiros, devido ao armazenamento das preferências e histórico do usuário. Recomendar oponentes também é significativo, pois aumenta a competitividade no servidor. Os jogadores que estão no mesmo nível de habilidade poderão jogar entre si. Os parceiros em jogos que mais obtiveram vitórias podem ser recomendados para jogarem juntos novamente, já que teoricamente eles possuem um entrosamento maior.

Referências

BLANCHETTE, J., AND SUMMERFIELD, M. 2008. *C++ GUI Programming with Qt 4*, 2 ed. Prentice Hall.

GUEDES, A., SANTOS, D., DO NASCIMENTO, J., SALES, L., PERKUSICH, A., AND ALMEIDA, H. 2008. Set your multimedia application free with brisa framework: An open source upnp implementation for resource limited devices. In *Proceedings of the 5th IEEE Consumer Communications and Networking Conference*, 1257–1258.

SALES, T. B. M., SALES, L. M., PEREIRA, M. F., OLIVEIRA, H. A., PERKUSICH, A., AND JUNIOR, M. A. S. 2008. Towards the upnp-up: Enabling user profile to support customized services in upnp networks. In *Proceeedings of the Second International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies*, 206–211.

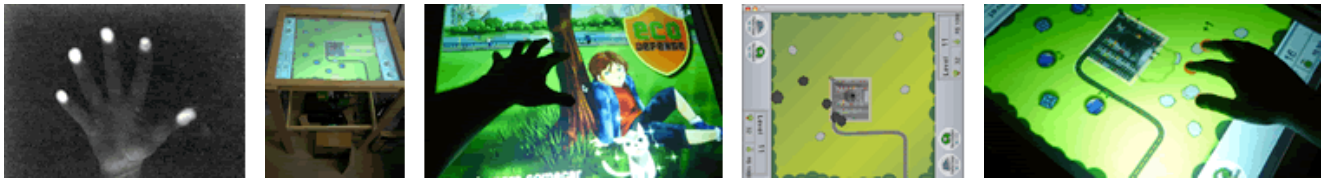
WEISER, M. 1991. The Computer for the 21st Century. *Scientific American 265*, 3 (Sept.), 66–75.

[10] http://maemo.org/

[11] http://wiki.enlightenment.org/index.php/Edje

Construção de um jogo multiusuário para uma superfície de projeção multitoque

Pedro G. Brandão Saulo C. R. Braga Carla D. Castanho Ricardo P. Jacobi
Departamento de Ciência da Computação, Universidade de Brasília, Brasil



(a) (b) (c) (d) (e)

Figura 1: *Jogo* Eco Defense *desenvolvido para uma mesa multitoque.*

Resumo

Este artigo tem como principal objetivo apresentar o jogo eletrônico, denominado *Eco Defense*, construído para uma superfície horizontal sensível a múltiplos toques. Assim, este trabalho aborda um paradigma de interação mais natural entre jogador e o sistema utilizando o toque das mãos. Considerando a complexidade e a dinamicidade inerentes do software de um jogo eletrônico, utilizou-se uma arquitetura de software baseada nos conceitos de engenharia de software já estabelecidos, como sistema de componentes e padrões de projeto. Devido à ausência de soluções multitoque acessíveis no mercado, o trabalho envolveu, também, a construção da mesa multitoque com dispositivos e materiais de baixo custo para ser utilizada como interface para jogo.

Keywords: jogo eletrônico, interfaces naturais, multitoque, arquitetura de software, sistema de componentes

Author's Contact:

{pedro, saulo}@beholdstudios.com.br
{carlacastanho, rjacobi}@cic.unb.br

1 Introdução

Para o homem, interagir com objetos sobre uma mesa é considerado bastante natural. Fazendo uma analogia ao computador, uma interface gráfica com a possibilidade de arrastar e manipular documentos e programas com os próprios dedos torna-se mais natural. A naturalidade do toque também pode ser empregada em jogos eletrônicos. Sabe-se que, a manipulação de elementos e objetos em um jogo tradicional de tabuleiro torna o mesmo significativamente interessante e envolvente. Da mesma forma, a possibilidade de interagir com um jogo eletrônico através do toque em uma superfície, traz um novo conceito de jogabilidade e interatividade.

É possível encontrar diversos trabalhos, [Corso et al. 2008], [Bomark 2007], [Shen et al. 2004], [spa 2009], [ver 2009], cujo foco é o estudo de interfaces de toque em sistemas multimídia interativos. Nestes trabalhos, jogos tradicionais como *Tangram* e *Game of Life*, por exemplo, foram desenvolvidos para superfícies multitoque com o objetivo de promover uma interação mais natural do jogador com o jogo.

Neste contexto, propõe-se um jogo eletrônico, denominado *Eco Defense*, cujo mecanismo de *input* é uma superfície horizontal sensível a múltiplos toques, onde diversos usuários podem simultaneamente manipular objetos virtuais.

Um dos desafios que envolvem o emprego de interfaces naturais, como o toque por exemplo, é a obtenção do equipamento que viabiliza o projeto. Devido a ausência de soluções prontas que atendam os requisitos técnicos e econômicos, este trabalho envolve, também, a construção da mesa multitoque a ser utilizada pelo jogo proposto. Esta proposta emprega técnicas de visão computacional para o reconhecimento dos toques realizados sobre a mesa.

Além disso, o desenvolvimento de um jogo eletrônico requer o entendimento e a aplicação dos conceitos de Engenharia de *Software* dentro deste cenário específico. O alto grau de interação e comunicação das entidades do jogo entre si e com o *hardware* influenciam na complexidade destes projetos. Dessa forma, o desenvolvimento do jogo proposto neste trabalho fundamenta-se nos padrões e conceitos de engenharia de *software* já estabelecidos, tais como arquitetura de sistema, camadas de abstração, reusabilidade de código e padrões de projeto [Doherty 2003].

2 Trabalhos correlatos

Os resultados publicados por [Bomark 2007] indicam que com o avanço da performance dos computadores pessoais e a disponibilidade de câmeras eficientes, é possível realizar processamento de imagens em tempo real para a construção de interfaces multitoque, e ainda preservar poder computacional para aplicações gráficas intensas, como jogos e interfaces de usuário atrativas.

Um projeto desenvolvido na *Lulea University of Technology*, na Suíça, propõe a construção de uma mesa multitoque com uma superfície de acrílico de $6mm$ para refletir uma luz infravermelha, através de um fenômeno físico denominado reflexão interna total frustrada (FTIR—*Frustrated Total Internal Reflection*) [Bomark 2007]. Quando o usuário pressiona seu dedo contra a superfície de acrílico, a reflexão total da luz dentro da superfície é quebrada naquele ponto, iluminando o dedo. Desta forma, uma câmera que filma a superfície é capaz de detectar a luz infravermelha naquele ponto e assim um *software* pode processar a imagem a procura destes pontos acesos.

O *Music Technology Group*, na *Universitat Pompeu Fabra* em Barcelona, propôs uma aplicação de uma mesa multitoque que consiste em um instrumento musical inovador, denominado *reacTable* [Jordà et al. 2005]. Seu principal componente é uma superfície tangível. A *reacTable* trabalha principalmente com marcadores fiduciais conhecidos pelo sistema. À medida que objetos são movimentados na superfície, ocorre um *feedback* sonoro e visual, de acordo com o tipo de fiducial. Certos tipos de objetos interagem entre si, sendo possível que a rotação ou a distância influencie seus efeitos sonoros.

Uma das principais formas de interação homem-máquina da atualidade é a utilizada em jogos eletrônicos. Hoje, diversos componentes fazem parte do computador ou *console* a fim de maximizar a performance dos jogos eletrônicos e melhorar sua jogabilidade. Um elemento importante no avanço da jogabilidade destes dispositivos é a inovação nos mecanismos de *input*. Diversos trabalhos de pesquisa e produtos comerciais inovam neste conceito, com multitoque, acelerômetros e câmeras de vídeo. De acordo com pesquisadores da *Brown University* [Katzourin et al. 2006], muitos dos jogadores se surpreenderam com a facilidade de aprendizado e naturalidade destes novos mecanismos de interação.

Estudantes do Centro de aplicações em Realidade Virtual da *Iowa State University*, nos Estados Unidos, desenvolveram uma mesa multitoque e um jogo multijogador interativo no âmbito do projeto denominado *Sparsh UI* [spa 2009]. Trata-se de jogo em

que dois times, situados em lados opostos da mesa, devem simultaneamente, através de toques sobre a superfície, atacar as bases do time oposto ou defender suas próprias instalações virtuais. Este tipo de trabalho põe em prática a discussão sobre multiusuários em superfícies multitoque, trazendo um novo paradigma de interação com as tecnologias.

Os trabalhos e projetos descritos nesta seção demonstram o direcionamento de esforços no desenvolvimento de jogos eletrônicos que incluam uma interação mais natural e envolvente para o usuário.

3 Reconhecimento de toques

O estudo e a pesquisa sobre a utilização de jogos em mesas multitoque demanda conhecimentos a respeito deste novo paradigma, em particular as técnicas voltadas para reconhecimento de toques através da captura de imagens. Uma das soluções existentes para reconhecimento de toques em uma superfície, utiliza um computador dotado de visão computacional com a habilidade de interpretar o posicionamento dos dedos. Uma câmera conectada ao computador captura imagens dos dedos dos usuários sobre a superfície multitoque. Através de algoritmos de processamento de imagens, o posicionamento de cada dedo e seus movimentos são calculados e transmitidos para a aplicação.

A biblioteca utilizada neste trabalho para o processamento das imagens capturadas através da câmera é a *Touchlib* [tou 2009]. Foi desenvolvida especificamente para a criação de superfícies com interação multitoque. Seu funcionamento consiste na aplicação de filtros de forma customizável que permite uma maior adequação às necessidade do projeto. Para a execução destes filtros, a *Touchlib* faz uso extensivo da *OpenCV* (*Open Source Computer Vision Library*) [ope 2009], uma biblioteca multiplataforma para desenvolvimento de aplicativos de processamento de imagens.

A *Touchlib* atua de acordo com o paradigma servidor–cliente, sendo que esta faz o papel de servidor e a aplicação faz o papel do cliente. A comunicação com este *software* é realizada usando o protocolo TUIO (*Tangible User Interface System*) [Kaltenbrunner et al. 2005], que foi desenvolvido especificamente para a transmissão do estado de objetos tangíveis e eventos de toque em uma superfície. Este opera sobre a camada UDP (*User Datagram Protocol*) de transporte, permitindo a divisão de aplicativos em mais de um computador.

4 Arquitetura de *software* aplicada a jogos eletrônicos

Atualmente, os jogos eletrônicos estão atingindo um alto grau de sofisticação e, como consequência, os *softwares* necessários para sua implementação têm se tornado cada vez mais complexos. As equipes de programação de jogos estão se tornando maiores e seus membros realizam tarefas cada vez mais especializadas. Além disso, de acordo com [Doherty 2003], a área de desenvolvimento de jogos está crescendo e amadurecendo ao ponto que as expectativas são similares às outras áreas de desenvolvimento de software. Assim, é essencial que jogos tenham uma arquitetura bem-definida para auxiliar, ou até mesmo viabilizar, o processo de desenvolvimento, manutenção e eventual evolução.

A arquitetura de um jogo deve contemplar a execução de diversas tarefas, tais como receber *input* do jogador, verificar colisão entre objetos, atualizar entidades do jogo, renderizar texturas e executar simulações. Embora tratem de aspectos diferentes do jogo, muitas destas tarefas devem ser executadas a cada *loop* iterativo. Esta arquitetura de *software* deve ser projetada de modo a evitar o forte acoplamento entre estas tarefas distintas, uma vez que reduz a dependência entre diferentes módulos do sistema, permitindo assim um maior reuso e modificações menos onerosas no processo de desenvolvimento.

4.1 Sistema de componentes

Conforme sugerido por [Folmer 2007], uma entidade de um jogo eletrônico é formada por diversos componentes distintos, cada qual com um comportamento específico. Estes componentes podem ser reutilizados por diferentes entidades, evitando assim o replicamento do código de forma desnecessária. Sabe-se que um sistema orientado a objetos não obriga a utilização de hierarquia de classes para estruturar as entidades do jogo. Logo, é possível utilizar uma alternativa que permite uma maior dinâmica da estrutura ainda em tempo de execução. Uma arquitetura orientada a componentes é uma solução que flexibiliza os comportamentos das entidades do jogo através de uma agregação de componentes independentes em uma classe única.

De acordo com [Rabin 2005], uma arquitetura de jogo baseada inteiramente em hierarquia de classes possui algumas limitações. O primeiro problema é o forte acoplamento entre classes. Um *software* com um acoplamento forte é indicativo de ser pouco modularizado e complexo de ser modificado. A herança cria o maior acoplamento possível entre duas classes, pois a classe derivada possui conhecimento sobre as estruturas públicas e protegidas da classe pai. Ou seja, mudanças na classe pai frequentemente representam mudanças na classe derivada.

Outra limitação, exposta em [Rabin 2005], é a falta de flexibilidade da hierarquia de classes. Em um sistema complexo, como um jogo eletrônico, podem existir situações difícies de modelar através de hierarquia. Existem características comuns entre entidades de jogo que não possuem herança comum, logo isto leva a uma replicação de código e comportamento semelhantes, prejudicando a manutenção desta estrutura.

Por fim, a estrutura imposta por hierarquias é estática em linguagens comuns como C++ e JAVA. Durante a execução não é possível alterar a herança de classes. Em um *software* comum isto não costuma ser um limitante, mas em um jogo eletrônico muitas vezes entidades alteram drasticamente seu comportamento em tempo de execução.

Um sistema de componentes utiliza composição para estruturar as entidades do jogo. Ao invés de cada entidade do jogo possuir uma classe distinta, apenas uma classe é criada com o objetivo de simbolizar todas as entidades do jogo. Esta classe pode, por exemplo, ser denominada de `GameEntity`. Esta classe única contém diversos componentes que juntos compõem a estrutura de uma entidade.

Pelo fato de cada componente ser independente dos demais, é possível que a entidade do jogo seja composta por qualquer combinação entre som, física, IA, GUI, gráfico, rede ou componentes de lógica de jogo. Nesta arquitetura de sistema, a entidade `GameEntity` não precisa conhecer a estrutura específica de cada componente, pois ela é apenas uma composição entre estas classes. Assim, o código mantém um acoplamento baixo, permitindo sua manutenção e modificação com mais facilidade.

5 *Game design* do *Eco Defense*

O jogo desenvolvido neste trabalho, denominado *Eco Defense*, tem como objetivo utilizar-se de uma mesa multitoque e trazer como conteúdo uma temática relacionada ao combate dos problemas do meio-ambiente. Durante o jogo, seus participantes deverão colaborativamente destruir a poluição gerada por uma fábrica no centro da tela. Os jogadores devem pressionar seus dedos contra estes poluentes, a fim de destruí-los, antes que atinjam e destruam a floresta ao redor da fábrica.

A destruição dos poluentes gera uma pontuação na forma de dinheiro virtual para os jogadores. Este dinheiro pode ser utilizado para construir torres que servem para auxiliar na eliminação dos poluentes. Os usuários tomam decisões em conjunto de como investir o dinheiro arrecadado pela destruição dos poluentes na construção de novas torres. À medida que o jogo avança, mais poluentes aparecem da fábrica e mais torres podem ser construídas.

Quanto mais jogadores entram no jogo, maiores são as chances de se atingir um melhor resultado. Com mais dedos disponíveis, é possível destruir mais poluentes ao mesmo tempo e obter mais dinheiro para investimento. Assim o jogo promove a participação de vários usuários colaborativamente.

A interface do jogo *Eco Defense* deve atender aos requisitos da mesa multitoque, sendo distribuída em torno da superfície de projeção. Desta maneira todos os usuários podem vivenciar a mesma experiência. A interface é replicada em lados opostos da

projeção, e contém os valores referentes aos pontos de vida da floresta, dinheiro arrecadado e o nível de dificuldade atingido pelos jogadores.

6 Construção da mesa multitoque

Através do estudo dos diversos trabalhos correlatos e utilizando-se do aprendizado dado pelas soluções previamente testadas, foi possível construir uma mesa multitoque a baixo custo com a utilização de componentes e equipamentos facilmente encontrados no mercado. A estrutura da mesa está ilustrada nas Figuras 1(b) e 2.

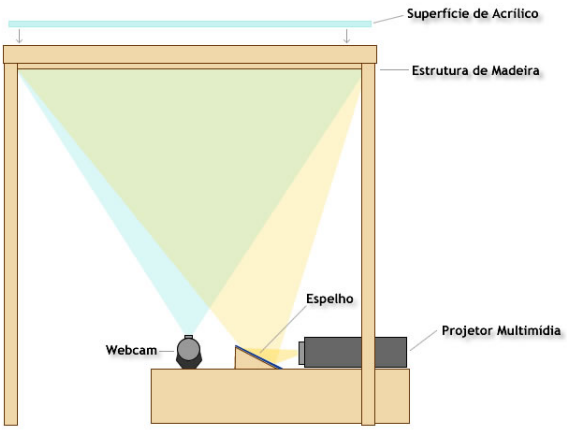


Figura 2: *Estrutura da mesa multitoque.*

Sua estrutura é constituída de madeira que dá suporte à superfície e se posiciona a uma altura aproximada de 1 metro, para que os usuários possam interagir ainda em pé. A superfície é constituída de uma placa de acrílico de espessura de $5mm$ e dimensões aproximadas de $65x45cm$. A espessura foi determinada para garantir a sustentação da superfície mesmo quando pressionada por vários usuários.

A projeção da imagem é obtida através de um projetor XVGA de resolução padrão $1024x768$ *pixels*, a uma distância aproximada de $100cm$ da superfície com auxílio de um espelho. Abaixo da superfície, utiliza-se uma película de papel vegetal para receber a imagem gerada pelo projetor. Esta superfície é essencial, também, para a iluminação correta dos dedos sobre a superfície. Sem a película, toda a mão do usuário seria iluminada igualmente. Com a película, os dedos ficam melhor iluminados se comparados à palma da mão.

Foi utilizado um sistema de iluminação direta, assim, a estrutura serve de apoio para posicionar o acrílico a uma distância adequada de duas fontes de luz infravermelhas posicionados abaixo da mesa junto ao projetor. A iluminação é oriunda de 96 LEDs IR montados sobre duas placas de circuito. Estes LEDs são ligados a uma fonte ATX de 12V e organizados em 12 circuitos paralelos de 8 LEDs. Os LEDs infravermelhos utilizados possuem uma voltagem (*forward voltage*) de $1,4V$ e necessitam de uma corrente aproximada de $20mA$. Para tal, utilizou-se um resistor de $39/ohm$ em cada circuito paralelo.

Por fim, com o objetivo de reduzir os gastos, utilizou-se uma câmera comum, *Microsoft VX-1000*. Esta câmera possui características adequadas de resolução (320x240 *pixels*) e *frame rate* ($60Hz$), e permite a remoção do filtro de luz infravermelho de fábrica. Foram incluídos três filtros de luz visível dentro da câmera, que melhoram o contraste da imagem com relação a luz infravermelha. Como filtro, é utilizado filme fotográfico revelado exposto à luz visível que possuem esta característica de filtragem, fornecendo resultado semelhante à figura 1(a).

7 Implementação do *software* interativo

Neste trabalho, em toda implementação do jogo (cujo resultado pode ser visto nas figuras 1(c), 1(d) e 1(e)), foi utilizada a linguagem de programação C++. De acordo com [Rabin 2005], esta linguagem é amplamente usada nos jogos comerciais, sendo a mais adotada para tal finalidade. Utilizou-se também uma biblioteca multiplataforma denominada SDL (*Simple DirectMedia Layer*) [sdl 2009]. Uma de suas vantagens é oferecer uma fina camada entre o *hardware* e *software* para prover serviços multimídia.

Com relação à parte gráfica, a SDL foi utilizado apenas para disponibilizar o acesso direto ao ambiente *OpenGL* (*Open Graphics Library*). Para carregar imagens em diversos formatos foi utilizada a pequena biblioteca *SOIL* (*Simple OpenGL Image Library*) [soi 2009], que carrega imagens diretamente em texturas *OpenGL*. Para carregar fontes *TrueType* e gerar texturas com texto foi usada a FTGL (*Free TrueType OpenGL Library*) [ftg 2009], que por sua vez usa a biblioteca *FreeType*.

Para a parte de áudio, a abstração da SDL fornece acesso direto ao *stream* de áudio. Assim, foi necessário a utilização do SDL_mixer, um *plugin* para a SDL que permite a execução de diversas faixas simultaneamente. Por fim, para o *input* foi utilizada a *Touchlib*. Do ponto de vista do *software*, este opera de acordo com o protocolo TUIO, responsável pela interface entre *Touchlib* e a aplicação do jogo.

7.1 Visão geral da arquitetura

A arquitetura do jogo é composta por uma controladora principal `Engine` que gerencia os subsistemas e as entidades de jogo. A classe `Engine` é intrinsecamente bastante simples. Ela gerencia as fases de inicialização, execução e encerramento do sistema.

Durante a inicialização, instancia todos os subsistemas e, durante a rotina de encerramento, finaliza cada um na ordem inversa. A fase de execução consiste em um *loop* infinito que atualiza cada subsistema e todos os `GameObjects`. Este *loop* somente é encerrado quando um dos subsistemas solicita término de execução.

7.2 Subsistemas

A arquitetura implementada foi dividida em quatro subsistemas: Gráfico, *Input*, Som e Eventos. Os subsistemas acessam uma controladora de mensagens, denominada `EventManager` do subsistema Eventos, para se comunicar e receber eventos de outros subsistemas.

A arquitetura do jogo implementado seguiu o padrão de sistema de componentes. Assim, além de uma controladora principal, todo subsistema possui famílias de componentes que agregam comportamento às entidades do jogo. Os componentes possuem nomes com prefixo `GOC`, como em `GOCRenderTexture`, componente do subsistema gráfico.

7.2.1 Gráfico

O subsistema gráfico possui uma controladora principal denominada `Graphics`. Esta é responsável pela renderização de todos os elementos visíveis na tela. Cada componente da família `GOCRender` se registra na controladora para serem exibidos na tela. A atualização do `Graphics` consiste em limpar o *buffer* de exibição e executar a atualização de todos os componentes registrados para este subsistema. Cada componente `GOCRender` registrado é desenhado neste *buffer*.

7.2.2 *Input*

O subsistema de *input* é controlado primariamente por uma classe denominada `Input`. A entrada do jogo tem duas fontes, uma é oriunda da SDL e trata de convencionais eventos de teclado e mouse, e outra oriunda da *Touchlib*, que trata eventos transmitidos por meio do protocolo TUIO. Um cliente fornecido em C++, denominado `TuioClient`, se associa a uma porta UDP pré-estabelecida, escutando os eventos que nela chegarem. Esta classe é executada em outra *thread*, permitindo a escuta dos eventos de forma paralela. Assim, a classe *Input* recebe os eventos do `TuioClient` e os repassa para a gerenciadora de eventos do jogo.

Qualquer entidade do jogo interessada em receber eventos de *input* deverá registrar-se como receptora do tipo de evento `FingerInputEvent`. Cada evento possui posição e um ID, referente à "instância do dedo" que provocou aquele evento. Desta forma, é possível acompanhar o movimento ao longo da superfície de um único dedo com facilidade.

7.2.3 Áudio

O subsistema responsável pelo áudio é controlado pela classe `Sound`. Durante a inicialização, é aberta uma instância do *SDL_mixer*. Como a execução do som é delegada ao *SDL_mixer*, não é necessário nenhum passo de atualização. Além disso, compõem este subsistema componentes da família `GOCSound`, que são responsáveis por carregar e reproduzir as faixas de áudio.

7.2.4 Eventos

A comunicação entre objetos é essencial ao sistema de qualquer jogo. Existem casos em que o fluxo de informação é muito complexo e um mecanismo mais flexível deve ser adotado. Para o caso em que várias entidades estão interessadas na notificação da ocorrência de determinado tipo de evento, ou quando se deseja enviar um evento mas não se sabe previamente quem deve recebê-lo, utiliza-se um sistema de mensagens.

Existem diversos tipos de evento. Para cada tipo, cria-se uma classe que herda de `Events`, que pode conter todas as estruturas de dados necessárias. O `EventManager`, única classe *singleton* [Gamma et al. 1995], é responsável pelo gerenciamento dos eventos. Objetos que desejam receber certo tipo de evento devem se registrar com esta classe, informando também o tipo de evento pelo qual se interessa. Objetos que desejam enviar um evento bastam especificar o evento como parâmetro na chamada de `SendEvent(Event)` do `EventManager`. Em seu funcionamento interno, a classe `EventManager` utiliza a estrutura de dados `std::map`, juntamente com `boost::signals`. O envio de eventos pode ser instantâneo ou pode ser atrasado (*delayed*). Neste segundo caso, o evento é enviado apenas no início do próximo *update*.

7.3 *Game Objects* e *Game Object Components*

Um *Game Object* representa uma entidade que pode existir no mundo do jogo. Consiste de uma identificação, um *Transform* (representando a posição, orientação e escala do objeto), e um conjunto de GOCs (*Game Object Components*)[Stoy 2006]. Essencialmente qualquer entidade do jogo é um *game object*, inclusive entidades não visíveis. Em geral, deseja-se sempre ter componentes associados a um *game object*, caso contrário este não terá comportamento algum.

O objetivo de cada GOC é oferecer interfaces simples e funcionalidades muito específicas, permitindo que os *game objects* possam fazer uso de diversos GOCs de forma flexível. Um GOC abstrato descreve uma interface comum que permite com que cada *Game Object* possa gerenciar seus próprios GOCs. Componentes são organizados de acordo com suas funcionalides. Estes grupos são denominados famílias. Cada componente obrigatoriamente pertence a uma família, que define a interface comum aqueles tipos de componentes.

O gerenciamento de componentes em *game objects* consiste em adicionar e obter um componente. Sendo assim, a estrutura de dados adotada é o `std::map` da STL (*Standard Template Library*), onde é feita a relação entre `ComponentIDs` e `Component*`. Para obter um componente de determinada família é usado o método `getGOC` especificando o ID da família desejada. Para adicionar um GOC, basta passar o ponteiro para o GOC que este será adicionado ao mapa de componentes.

8 Conclusão

O desenvolvimento de jogos eletrônicos tem se tornado, cada vez mais, uma área importante da engenharia de software. A evolução de sistemas interativos leva a um estudo aprofundado de diversas técnicas e conceitos de arquitetura, padrões e processos. Sem este estudo, os jogos complexos de hoje se tornariam inviáveis, pois alguns ultrapassam milhões de linhas de código, demandam equipes com dezenas de profissionais e algoritmos muito eficientes. [Rabin 2005]

Para acompanhar tal evolução, a pesquisa e o estudo de novas interações homem-máquina são essenciais. Através do estudo e uso de novas tecnologias se rediscute os paradigmas de interação entre jogadores e jogos eletrônicos. Assim, a imersão, o envolvimento e o entretenimento atingem outros patamares.

Neste trabalho foi apresentado o jogo eletrônico *Eco Defense*, desenvolvido para uma superfície multitoque. O resultado obtido demonstrou a possibilidade de inovação através da utilização de conceitos já estabelecidos por diversos estudos. Além disso, a abordagem de desenvolvimento de software adotada promove a discussão sobre a adequação dos processos de arquitetura de sistemas para jogos eletrônicos. Por fim, o trabalho demonstrou a possibilidade de construção de interfaces naturais a partir de elementos de baixo custo disponíveis no mercado.

Referências

BOMARK, P. 2007. *Visualization and Prototyping of a Multitouch Display Device*. Master's thesis, Luleå tekniska universitet.

CORSO, J., YE, G., BURSCHKA, D., AND HAGER, G. 2008. A practical paradigm and platform for video-based human-computer interaction. *Computer, IEEE Computer Society 41*, 5, 48–55.

DOHERTY, M. 2003. A software architecture for games. *University of the Pacific Department of Computer Science Research and Project Journal 1*, 1.

FOLMER, E. 2007. Component-based game development. *Lecture Notes in Computer Science 2007*, 66–73.

2009. FTGL: A font rendering library for OpenGL. `http://homepages.paradise.net.nz/henryj/code/`. Acessado em junho/2009.

GAMMA, E., HELM, R., JOHNSON, R., AND VLISSIDES, J. 1995. *Design Patterns*. Addison Wesley.

JORDÀ, S., KALTENBRUNNER, M., GEIGER, G., AND BENCINA, R. 2005. The reactable*. In *Proceedings of the International Computer Music Conference (ICMC 2005)*.

KALTENBRUNNER, M., BOVERMANN, T., BENCINA, R., AND CONSTANZA, E. 2005. Tuio: A protocol for table-top tangible user interfaces. In *Proc. of the 6th International Workshop on Gesture in Human-Computer Interaction and Simulation*.

KATZOURIN, M., IGNATOFF, D., QUIRK, L., JR., J. L., AND JENKINS, O. 2006. Swordplay: Innovating game development through vr. *Computer Graphics and Applications, IEEE 26*, 6, 15–19.

2009. OpenCV: Open Source Computer Vision. `http://opencvlibrary.sourceforge.net/`. Acessado em junho/2009.

RABIN, S., Ed. 2005. *Introduction to Game Development*. Charles River Media.

2009. SDL: Simple directmedia layer. `http://www.libsdl.org`. Acessado em junho/2009.

SHEN, C., VERNIER, F., FORLINES, C., AND RINGEL, M. 2004. Diamondspin: An extensible toolkit for around-the-table interaction. In *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, 167–174.

2009. SOIL: Simple OpenGL Image Library. `http://www.lonesock.net/soil.html`. Acessado em junho/2009.

2009. Sparsh UI: A multitouch api for any multitouch hardware / software system. `http://code.google.com/p/sparsh-ui/`. Acessado em junho/2009.

STOY, C. 2006. *Game Programming Gems 6*. Charles River Media, ch. Game Object Component System.

2009. Touchlib: a library for creating multi-touch interaction surfaces. `http://nuigroup.com/touchlib/`. Acessado em junho/2009.

2009. Verve: Veneral purpose agents. `http://verve-agents.sourceforge.net/`. Acessado em junho/2009.

Descoberta de Posições Táticas em Ambientes Virtuais com a Utilização de Agrupamentos Espaço-Temporais

Ivan Medeiros Monteiro
Instituto de Informática
UFRGS

Vania Bogorny
Departamento de Informática e Estatística
UFSC

Luis Otavio Alvares
Instituto de Informática
UFRGS

Resumo

A demanda por comportamentos convincentes em jogos de computador tem orientado a pesquisa em inteligência artificial com o objetivo de aprimorar o comportamento dos personagens em jogos. A fim de alcançar um comportamento convincente, os personagens devem usar o conhecimento sobre o mundo no qual atuam. A representação do ambiente é muito importante para executar um comportamento mais sofisticado, mas na maioria dos jogos essa representação ainda continua sendo definida manualmente de forma estática pelo projetista do ambiente, levando os personagens a um comportamento previsível e pouco convincente. Apesar da geração automática de posições táticas estar sendo tratada por alguns trabalhos, não existe uma solução geral e as abordagens atuais possuem muitas limitações. Em função disto, este trabalho apresenta alguns resultados preliminares do uso de agrupamento espaço-temporal para encontrar posições táticas em ambientes virtuais, permitindo que os *bots* se adaptem ao modo como as partidas são jogadas.

Palavras-chave: Agrupamentos Espaço-Temporais, Mineração de Dados, Sistemas Multiagentes, Inteligência Artificial.

Contato:

{immonteiro,alvares}@inf.ufrgs.br
vania@inf.ufsc.br

1 Introdução

Na maioria dos jogos, informações táticas [1] são especificadas no ambiente pelo projetista do mundo virtual. Entretanto, a especificação dessas regiões interessantes durante a criação dos mapas é um trabalho que consome muito tempo. Em geral, essa tarefa é realizada manualmente, o que pode limitar as interações no ambiente, dado que algumas localizações interessantes podem não ser identificadas pelo projetista.

Algumas sequências de ações simples, como ações lineares predefinidas, podem ser orquestradas com sucesso pelo projetista do ambiente através da especificação de posições táticas. Entretanto, seu trabalho leva em consideração apenas o ambiente estático, e não avalia as interações entre personagens no ambiente. Além disso, a especificação manual de posições táticas pode ser um trabalho pesado, sendo suscetível a falhas, dado que o projetista pode não prever todos os pontos interessantes que podem surgir no ambiente durante as partidas.

A situação é um pouco mais crítica quando os mapas são gerados automaticamente, porque o projetista não pode espeficificar diretamente essas regiões. Uma solução para esse tipo de problema é a descoberta automática dessas regiões interessantes a partir das experiências dos *bots* no ambiente. Isto permitiria aos *bots* ter não apenas um melhor conhecimento sobre o ambiente no qual estão inseridos, como também habilitaria o agente a aprender sobre os comportamentos e as estratégias dos oponentes.

Existem alguns trabalhos que abordam o problema das posições táticas a partir de diferentes aspectos. Alguns deles descrevem uma representação espacial das localizações interessantes [Tozour 2001] [Isla 2006], enquanto outros provêm uma forma de gerar automaticamente essas posições táticas [Tozour 2004] [Straatman et al. 2006] [Darken and Paull 2006] [Dill and Sramek 2004]. Entretanto, nenhum deles realiza a descoberta de localizações táticas usando o conhecimento obtido nas partidas.

[1] ex. regiões de batalhas, áreas de *camper*, área de comércio, etc

Assim, esse trabalho apresenta um caminho inicial para o uso dos métodos de descoberta de conhecimento baseado em agrupamento espaço-temporal, usando informações sobre a posição dos personagens para aprender qual tipo de evento ocorre em quais partes do ambiente. Nós também examinamos como os personagens se comportam em uma partida a fim de descobrir quais estratégias eles estão utilizando, através da análise de suas trajetórias no ambiente. Assim, conseguimos inicialmente descobrir locais perigosos e posições de *camper*[2] usando os dados gravados das partidas. O algoritmo de agrupamento por densidade, DBSCAN [Ester et al. 1996], é utilizado para encontrar as regiões perigosas enquanto o algoritmo CB-SMoT [Palma et al. 2008] é usado sobre as trajetórias dos personagens para descobrir as posições de *camper*.

Os experimentos foram executados com o jogo Unreal Tournament 2004, gravando as partidas através da interface GameBots [Kaminka et al. 2002]. O ciclo de aprendizado é realizado por partidas, onde ao final de cada uma, os logs são analisados a fim de atualizar o modelo do ambiente e para identificar como os personagens se comportam. Assim, é possível desenvolver *bots* adaptativos, que podem atuar de acordo com o modelo de mundo aprendido, contornando comportamentos repetitivos e tornando-se mais convincentes. Além disso, surgem novas oportunidades para o planejamento de caminho, levando em consideração os pesos que podem ser alocados para cada região descoberta.

O restante deste artigo está organizado da seguinte forma: a Seção 2 apresenta os trabalhos relacionados, a Seção 3 descreve brevemente as técnicas de mineração utilizadas, a Seção 4 mostra como realizar a descoberta de posições táticas utilizando agrupamento espacial e espaço-temporal, na Seção 5 são apresentados os experimentos realizados, e na Seção 6 se conclui este trabalho.

2 Trabalhos Relacionados

A representação de locais de interesse em jogos é importante para permitir que sejam desenvolvidos comportamentos convincentes para os personagens. Essa representação dá liberdade ao *bot* para que ele raciocine sobre as possíveis táticas a serem executadas no jogo. Diferentes tipos de jogos usam formas distintas de representar posições interessantes. Influence Maps [Tozour 2001] e Occupancy Maps [Isla 2006] são exemplos de representações comumente utilizadas em *RTS*, enquanto sistemas de *waypoints* são mais frequentemente utilizados em jogos de primeira e terceira pessoa.

Um Influence Map é uma representação espacial que armazena informações como: onde ocorreram batalhas importantes, quais regiões continuam inexploradas, ou onde o inimigo tem maior probabilidade de estar. Não existe um algoritmo único para a criação do Influence Map, e sua representação pode ser aplicada de diferentes maneiras. As diversas formas de se construir um Influence Map dependem das táticas e estratégias necessárias.

Os sistemas de *waypoints*, diferentemente do Influence Map, são usados principalmente para navegação através do ambiente. Eles são comumente representados por um grafo conexo e seus nós são posições no mundo. Assim, um sistema de *waypoint* representa todos os caminhos que os *bots* podem navegar através do ambiente. A fim de permitir comportamentos estratégicos, muitos desses nós são rotulados com informações táticas, criando uma representação espacial de locais interessantes, como um Influence Map faz.

Embora na maioria dos jogos as informações táticas sejam especificadas pelo projetista, existem alguns trabalhos que automatizam

[2] Personagem que espera em uma determinada posição o inimigo para uma emboscada.

a definição de pontos interessantes. Lars Lidén [Liden 2002] apresenta como o pré-processamento da relação entre os *waypoints* pode ser usado para gerar posições táticas para *bots* em jogos de primeira pessoa. Ele gera informação de visibilidade estática entre *waypoints*, pré-calculando se um *waypoint* tem visibilidade de outro. Usando visibilidade é possível determinar quais locais são potencialmente seguros e quais são perigosos. Entretanto, a principal limitação do trabalho de Lars é que a efetividade desse pré-calculo depende da capacidade do projetista em especificar a posição dos nós de forma apropriada. Em outras palavras, embora o projetista não precise definir os pontos táticos, ele ainda necessita definir as informações de caminhos no ambiente. Além disso, as informações táticas geradas são estáticas e previsíveis.

Alguns trabalhos propõem a análise de posições táticas em tempo de execução [Tozour 2004] [Straatman et al. 2006] [Darken and Paull 2006]. Paul Tozour propôs uma base de dados espacial criada em tempo de execução para permitir que personagens tomem decisões sobre ela [Tozour 2004]. Essa abordagem usa camadas de grids 2D para armazenar informações como área de ocupação, área de busca e tantas outras também interessantes. Entretanto, ela apresenta sérios problemas para tratar um ambiente 3D.

Baseada em sistemas de *waypoints*, a avaliação dinâmica de posições táticas [Straatman et al. 2006] usa informações estáticas e dinâmicas sobre o mundo para criar decisões táticas em tempo real. Ela usa a proximidade para selecionar *waypoints* e a visibilidade para pontuar essas posições, onde os maiores escores são os locais mais atrativos. O maior problema em usar essa avaliação em tempo de execução é devido ao grande número de operações custosas como *ray casts*[3]. Assim, muitas restrições devem ser feitas para tratar o problema de performance, como: limitar o número de posições avaliadas, restringir personagens para certas áreas do mapa e usar tabela de visibilidade pré-calculada.

Existem trabalhos voltados a encontrar um tipo específico de posições táticas, como posições de cobertura [Darken and Paull 2006], enfatizando a adaptabilidade ao ambiente dinâmico. Essa técnica combina anotações do ambiente com algoritmos de grade de sensores. A anotação do ambiente ainda é feita pelo projetista enquanto o algoritmo de grade de sensores testa as posições de cobertura por *ray cast* em tempo real. Assim, um conjunto de locais escondidos pode ser gerado dinamicamente.

De forma diferente, nossa abordagem descobre posições táticas com métodos baseados em agrupamento, minerando informações espaço-temporais. Nós estamos interessados em encontrar dois tipos distintos de posições táticas, que são as áreas de *camper* e as regiões de perigo. Assim, aplicamos duas diferentes técnicas para achar essas regiões.

3 Mineração de Dados

Neste trabalho, utilizamos técnicas de mineração de dados espaciais e espaço-temporais para a formação de agrupamentos que irão representar os pontos de interesse no ambiente. Um agrupamento é um grupo de dados que compartilha propriedades similares. Usualmente, cada agrupamento é um subconjunto de um conjunto inteiro de dados. O termo clusterização é bastante utilizado na comunidade de pesquisa para descrever métodos que agrupam dados não rotulados.

3.1 DBSCAN

DBSCAN *(Density-Based Spatial Clustering of Applications with Noise)* é um importante método de clusterização baseado em densidade. Ele foi desenvolvido por Ester [Ester et al. 1996] e teve influência de muitos outros métodos baseados em densidade. As principais vantagens do DBSCAN exploradas neste trabalho são: a capacidade de gerar agrupamentos de diferentes formas no conjunto de dados espaciais e a tolerância a dados ruidosos.

O DBSCAN procura por pontos de núcleo, que dão origem a um agrupamento, e tenta expandir o agrupamento inicial agregando os vizinhos mais próximos que atendem a algumas condições. Essas condições são baseadas em dois parâmetros: $MinPts$ e Eps. $MinPts$ é a medida de densidade que indica quantos pontos precisam estar na vizinhança para que tal ponto seja considerado pertencente a um agrupamento. Eps é a distância utilizada para delimitar a vizinhança. Assim, este método inicia um agrupamento a partir de um ponto núcleo, que possui pelo menos $MinPts$ vizinhos dentro de um raio Eps. Os agrupamentos são expandidos com a adição de novos pontos núcleos e seus limites são definidos pelos pontos de borda, pontos que estão na vizinhança de um ou mais pontos núcleos mas que não possuem vizinhos o suficiente em um raio Eps para serem considerados pontos núcleos. Os pontos que não estão na vizinhança de pontos de núcleo e que não atendem às condições impostas por $MinPts$ dentro de um raio Eps são considerados ruído.

A possibidade de criação de agrupamentos com diferentes formatos permite que o DBSCAN seja utilizado para descrever com maior precisão as posições táticas, que estão associadas a topologia do ambiente, que por sua vez não está restrita às formas pré-definidas utilizadas em outros algoritmos. A tolerância a ruído do método também é bastante importante na definição de posições táticas, dado que diversos eventos ocorrem por todo o cenário e é preciso descartar as regiões onde a densidade de ocorrência de determinados eventos é irrelevante.

3.2 CB-SMoT

CB-SMoT [Palma et al. 2008] é um método espaço-temporal de formação de agrupamentos aplicado sobre trajetórias brutas. Uma trajetória bruta é uma sequência de (tid, x, y, t), onde tid é o identificador do objeto móvel e x, y é a sua posição no espaço em um instante de tempo t. Uma trajetória bruta tem pouca ou nenhuma semântica. Neste trabalho, o algoritmo original foi alterado para suportar trajetórias no espaço tridimensional, ou seja, a trajetória bruta é uma sequência de (tid, x, y, z, t), onde o z é a terceira coordenada espacial.

O método CB-SMoT foi desenvolvido para transformar uma trajetória bruta em uma trajetória semântica, isto é, uma sequência de pontos importantes da trajetória do ponto de vista da aplicação. Os pontos importantes (subtrajetórias) correspondem a regiões de baixa velocidade. Os $clusters$ gerados são subtrajetórias com baixa velocidade média (abaixo de um parâmetro de velocidade denominado avg) e com duração maior ou igual a um parâmetro de tempo mínimo ($MinTime$).

O algoritmo CB-SMoT tem dois passos principais: inicialmente são computados os clusters de baixa velocidade e em um segundo passo estes $clusters$ são comparados com informações geográficas relevantes para a aplicação para atribuir semântica aos $clusters$. No presente trabalho, apenas o primeiro passo, a geração dos $clusters$, foi considerado, pois não há informações geográficas nos mapas utilizados.

A trajetória de um personagem é um dado espaço-temporal que possui a localização deste nos diferentes instantes de tempo durante um jogo. Os pontos da trajetória de um personagem são coletados, em intervalos de 200 milisegundos, a partir do log do jogo. Com o CB-SMoT é possível identificar quais são as partes lentas das trajetórias dos personagens, sendo esses trechos de lentidão associados com regiões de interesse no ambiente.

4 Geração de Posições Táticas

Em jogos, posições táticas são frequentemente usadas para representar posições defensivas, pontos de cobertura ou regiões de *camper*. O projetista de cenário normalmente marca essas posições no ambiente baseado no seu conhecimento sobre a dinâmica do jogo. É comum que essas localizações sejam representadas como anotações em *waypoints*, fazendo diferentes usos para o grafo de caminho, já que os *waypoints* são originalmente projetados para planejamento de caminho.

Essas informações táticas poderiam também se referir a posições de inventórios, onde armas e munição podem ser encontradas; locali-

[3] Verificação de interseção de objetos com um raio.

zação de kit-médico, onde é possível recuperar a vitalidade; pontos de reconhecimento, onde uma grande área é visível a partir dali; e outros pontos relevantes no ambiente. Esses pontos podem também representar locais que devem ser evitados, como áreas expostas ou locais de emboscadas. São os tipos de jogos e as estratégias que determinam as posições táticas nos jogos, podendo ser tanto negativas como positivas.

Embora as posições táticas possam ser associadas com *waypoints*, que são usados principalmente para navegação, algumas dessas localizações não são úteis como parte de um grafo de planejamento de caminho. Uma posição de *camper*, por exemplo, provavelmente não estará localizada em um caminho entre dois *waypoints* do ambiente, pois eles geralmente são o fim da linha. O mesmo ocorre com outras localizações, como pontos de cobertura. Assim, esses pontos não precisam necessariamente estar associados com os dados de procura de caminho, na realidade, o planejamento de caminho poderia ser mais eficiente se essas informações estivessem separadas.

A definição de todos os pontos de interesse no ambiente pode produzir um grande número de pontos, e isso é necessário para alcançar uma boa qualidade de comportamento. Embora localizações como cantos escuros e pontos de cobertura sejam facilmente identificadas por seres humanos, um projetista pode não detectar todos os pontos táticos interessantes e essa especificação manual pode conduzir os personagens a apresentar comportamentos pouco convincentes. Com isso, surge uma necessidade em definir automaticamente pontos táticos nos ambientes.

A partir do registro das ações de jogadores humanos é possível obter informações de localizações taticamente interesantes. Se um personagem se mantém numa mesma região por algum tempo ou se a mesma região é utilizada diversas vezes ao longo da partida, existe uma grande probabilidade dela ser uma região de algum interesse. Com o conjunto de regiões táticas candidatas, é possível avaliar, automaticamente, tanto a qualidade tática, como pontos de cobertura e pontos de emboscada, usando algoritmos de agrupamento espacial e espaço-temporal

4.1 Regiões Perigosas

É importante que os personagens consigam identificar quais regiões no ambiente são perigosas, para que eles possam evitá-las sempre que possível. As regiões com grande ocorrência de mortes são bastante críticas, pois normalmente representam áreas expostas que devem ser levadas em conta pelo personagem. Para identificar essas regiões levamos em consideração a densidade espacial das ocorrências de morte no ambiente.

Utilizamos o DBSCAN, um algoritmo de agrupamento espacial baseado em densidade, para identificar as regiões perigosas a partir da mineração sobre os dados espaciais de mortes dos personagens no ambiente. Duas características do DBSCAN são importantes para realizar essa tarefa. A primeira é a geração de agrupamentos de formatos variados (não-esféricos), o que permite uma representação adequada dentro do ambiente. A segunda é a possibilidade de filtrar ruídos, ignorando casos isolados de mortes para a formação de agrupamentos relevantes.

4.2 Pontos de Camper

Outro tipo importante de pontos que buscamos identificar neste trabalho são os pontos de *camper*. O objetivo dessa identificação é permitir que *bots* consigam executar comportamentos eficientes para a dinâmica do jogo e também consigam combater inimigos que se utilizam dessas regiões. Para isso, utilizamos o algoritmo CB-SMoT sobre a trajetória de cada personagem, identificando os pontos de baixa velocidade.

Uma característica dos pontos de *camper* é que eles são utilizados, geralmente, quando o personagem está parado ou movendo-se em baixa velocidade, já que ele se dirige para esses pontos onde aguarda o aparecimento de um inimigo. Por isso, buscamos identificar esses pontos de baixa velocidade observando a trajetória de cada personagem. Também é característica importante a ocorrência de muitas mortes originadas nesses pontos, por isso, também é interessante associar os pontos de baixa velocidade com as regiões onde são causadas as mortes no ambiente. Todavia, em nossa abordagem inicial utilizamos apenas os pontos de baixa velocidade nas trajetórias, o que efetivamente encontra os pontos de *camper*, mas poderia identificar também pontos onde um personagem tenha ficado preso no ambiente.

5 Experimentos e Avaliações

Os experimentos foram realizados com o jogo Unreal Tournament 2004, utilizando a interface GameBots [Kaminka et al. 2002] para a aquisição das informações do jogo. Foram realizadas partidas de 30 minutos em dois mapas distintos. Para a aplicação do DBSCAN, foram selecionados os dados relacionados com os eventos de morte no jogo, como pode ser visto na Listagem 1, em seguida esses dados foram convertidos para o formato de entrada do algoritmo, como pode ser visto na Listagem 2. Para o CB-SMoT o arquivo de entrada continha um conjunto de trajetórias representadas por (tid, x, y, z, t), sendo o tid a identificação da trajetória do *bot* e x, y, z a posição espacial do personagem no tempo t.

Listing 1: *Trecho de dados pré-selecionados para a identificação de regiões perigosas.*

```
KIL {Id CTF-FaceClassic.GBxPlayer1} {Killer CTF-
     FaceClassic.RemoteBot2} {DamageType XWeapons.
     DamTypeShockBeam}
IPLR {Id CTF-FaceClassic.RemoteBot2} {Name PogamutBot_0}
     {Location -954.83,265.28,-2160.03} {Rotation
     0,2171,0} {ManualSpawn False} {AutoTrace False} {
     Invulnerable False} {VisionTime 0.20} {ShowDebug
     False} {ShowFocalPoint False} {DrawTraceLines True}
IPLR {Id CTF-FaceClassic.GBxPlayer1} {Name Guedes} {
     Location 1720.93,832.10,-1651.04} {Rotation
     0,34786,0}
```

Listing 2: *Trecho do arquivo de entrada gerado para identificação de regiões perigosas.*

```
@RELATION location
@ATTRIBUTE x  REAL
@ATTRIBUTE y  REAL
@ATTRIBUTE z  REAL
@ATTRIBUTE class {winner,loser}
@DATA
1720.93,832.10,-1651.04,loser
-954.83,265.28,-2160.03,winner
```

Nestes experimentos, tanto o método baseado em densidade, quanto o método baseado em velocidade apresentaram resultados interessantes. O DBSCAN conseguiu encontrar as regiões do cenário que ofereceram maior perigo, a partir da concentração de morte. Vale ressaltar que o DBSCAN foi bastante tolerante a ruído, quando seus parâmetros foram definidos de forma adequada, apesar da identificação desses parâmetros não ter sido um processo imediato.

Nas Figuras 1 e 2, que representam os ambientes utilizados, foi possível perceber que o DBSCAN encontrou zonas de perigo (representadas pelo conjunto de regiões destacadas) em campo aberto e nas proximidades do *respawn*[4] de personagens, o que é bastante coerente para um jogo de primeira pessoa. O CB-SMoT também apresentou resultados muito interessantes, identificando os pontos de *camper* utilizados. Na Figura 3 foram encontrados pontos dentro das duas torres existentes no cenário e na Figura 4 foram encontrados pontos dentro de um pequeno bunker do cenário.

Em outras palavras, o DBSCAN conseguiu identificar as regiões de perigo como sendo aquelas que possuiam uma grande densidade de morte, levando em consideração a topologia do ambiente e descartando as regiões com mortes esporádicas. Já o CB-SMoT conseguiu identificar as posições em que eram realizadas emboscadas e onde ficavam os *campers*, isto porque a principal característica de um *camper* é aguardar o inimigo em posições específicas do cenário, o que pode ser identificado pelas subtrajetórias lentas de sua trajetória bruta.

Apesar dos bons resultados iniciais obtidos, a necessidade do ajuste manual dos parâmetros nos dois algoritmos foi o principal ponto

[4] Pontos onde ressurgem os personagens.

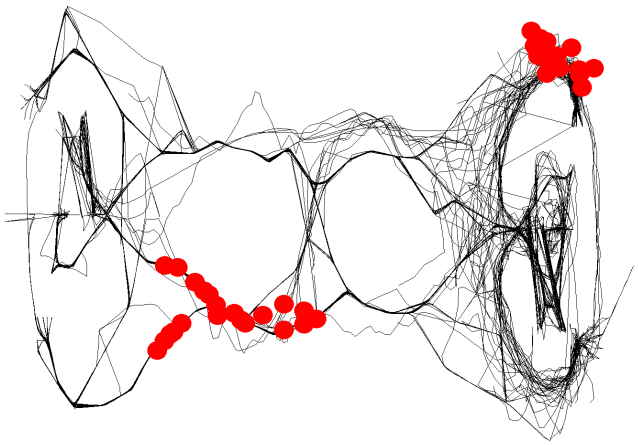


Figura 1: *Regiões perigosas descobertas no mapa FaceClassic com o DBSCAN (Eps=0.01, MinPoints=15).*

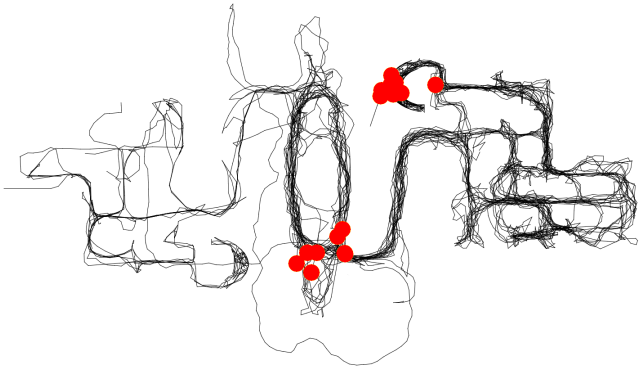


Figura 2: *Regiões perigosas descobertas no mapa December com o DBSCAN (Eps=0.01, MinPoints=6).*

negativo encontrado. No caso do DBSCAN, mesmo existindo trabalhos que analisam a estrutura de agrupamentos através da variação do Eps, os mesmos não foram utilizados nos experimentos. No caso do CB-SMoT, todos os parâmetros ainda precisam ser definidos manualmente, o que dificulta, mas não impossibilita, a identificação das regiões de interesse de forma totalmente automática.

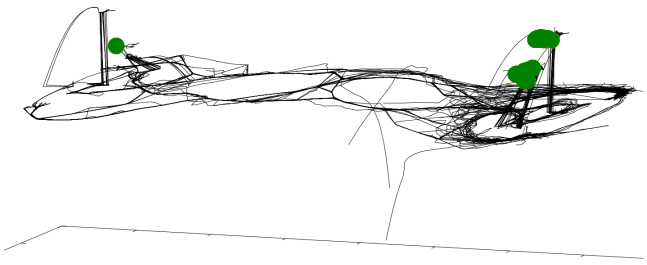


Figura 3: *Regiões de camper descobertas no mapa FaceClassic com o CB-SMoT (avg=0.1, minTime=12).*

6 Conclusões

A especificação de posições táticas dentro de ambientes virtuais é bastante importante para permitir comportamentos mais convincentes dos *bots*. Por isso, é interessante viabilizar métodos que permitam a definição desses pontos com a mínima interferência humana.

Com este trabalho concluimos que é viável realizar a descoberta automática de posições táticas em ambientes virtuais, utilizando técnicas de agrupamento espacial e espaço-temporal para diminuir o

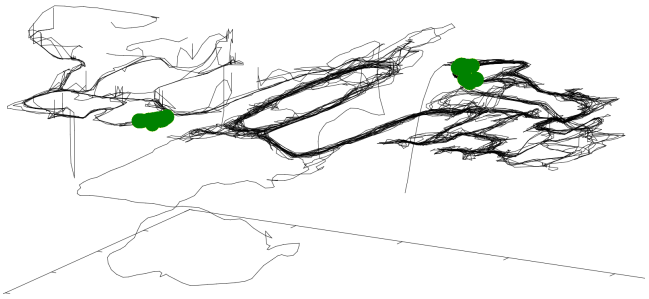


Figura 4: *Regiões de camper descobertas no mapa December com o CB-SMoT (avg=0.1, minTime=20).*

trabalho do projetista, encontrando algumas vezes regiões não previstas e pouco intuitivas. A aplicação dessas técnicas torna possível o desenvolvimento de *bots* adaptativos, que adquirem mais experiência sobre seu ambiente à medida em que jogam novas partidas. Assim, a aplicação do processo de descoberta de conhecimento sobre jogos é vista como uma área promissora, uma vez que diminui o esforço humano e aumenta o grau de aceitabilidade e convencimento sobre a inteligência dos *bots*. Alguns passos, entretanto, ainda precisam ser trilhados de modo a permitir que se diminua cada vez mais a intervenção humana na avaliação dos modelos gerados.

Dentre os principais trabalhos futuros podemos citar o estudo sobre o ajuste automático dos parâmetros dos algoritmos; a associação de pontos onde são causadas as mortes com os pontos de baixa velocidade na trajetória, para uma filtragem dos agrupamentos candidatos; e a utilização dessas informações para o raciocínio dos *bots*.

Referências

DARKEN, C. J., AND PAULL, G. H. 2006. Finding cover in dynamic environments. In *AI Game Programming Wisdom 3*, Charles River Media, 405–416.

DILL, K., AND SRAMEK, A. 2004. Performing qualitative terrain analysis in master of orion 3. In *AI Game Programming Wisdom 2*, Charles River Media, 391–398.

ESTER, M., PETER KRIEGEL, H., S, J., AND XU, X. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. AAAI Press, 226–231.

ISLA, D. 2006. Probabilistic target tracking and search using occupancy maps. In *AI Game Programming Wisdom 3*, Charles River Media, 379–388.

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.

LIDEN, L. 2002. Strategic and tactical reasoning with waypoints. In *AI Game Programming Wisdom*, Charles River Media, 211–220.

PALMA, A. T., BOGORNY, V., KUIJPERS, B., AND ALVARES, L. O. 2008. A clustering-based approach for discovering interesting places in trajectories. In *SAC*, 863–868.

STRAATMAN, R., BEIJI, A., AND VAN DER STERREN, W. 2006. Dynamic tactical position evaluation. In *AI Game Programming Wisdom 3*, Charles River Media, 389–404.

TOZOUR, P. 2001. Influence mapping. In *Game Programming Gems 2*, Charles River Media, 287–297.

TOZOUR, P. 2004. Using a spatial database for runtime spatial analysis. In *AI Game Programming Wisdom 2*, Charles River Media, 381–390.

Desenvolvimento de um Jogo de Corrida em FPGA

Italo S. Santos
João Carlos N. Bittencourt
Anfranserai M. Dias
Universidade Estadual de Feira de Santana

Abstract

This paper describes a racing game implementation using reconfigurable circuits in FPGA. The model uses as output interface the VGA monitors and manual controls using push-buttons. That guarantees a better debugging of developed circuits in FPGA, increasing its potentiality and interactivity with users. The differential of this project can be defined by the use of an integrated circuit of low cost, easy maintenance and programmed in Verilog hardware description language. This project is divided in two parts, first will be presented an approach about the content of the circuit and finally described the main parts of the developed model.

Keywords:: FPGA, VGA Interface, Verilog

Author's Contact:

italo.ecomp@gmail.com
joaocarlos@ecomp.uefs.br
anfranserai@ecomp.uefs.br

1 Introdução

O avanço tecnológico no desenvolvimento de microprocessadores cresceu de forma exponencial nas últimas décadas, tornando-os cada vez menores, e embarcando um número maior de transistores. Este ritmo de crescimento pode ser atestado a partir da Lei de Moore [Schaller 1997] e verificado pela inserção periódica no mercado de novos Circuitos Integrados (CIs) com grande capacidade de processamento. Esta lei é devida a Gordon E. Moore, que em 1965 observou que a densidade de componentes em circuitos integrados dobrava a intervalos regulares, inferindo que este comportamento perduraria por muito tempo ainda. O intervalo medido por Moore para que a densidade média de CIs dobrasse foi de 18 meses, o que ainda hoje permanece uma taxa estável [Calazans 1998].

Esta elevada taxa de crescimento resulta em um curto espaço de tempo para que um novo projeto chegue ao mercado. Aliado a isto, têm-se o aumento de complexidade dos projetos devido à alta densidade de componentes nos CIs, *System-on-Chip* (SoC), mostrando a necessidade de novas metodologias que auxiliem na construção. Uma alternativa para o desenvolvimento destes CIs, está no processo de criação de *IP-cores* baseado no RUP *(Rational Unifed Process)* com prototipação em dispositivos FPGA *(Field Programable Gate Array)*.

A utilização de FPGAs para o desenvolvimento de *IP-cores* é uma alternativa para projeto de circuitos integrados, pois permite ao desenvolvedor utilizar paradigmas de programação em alto nível, podendo testar e modificar o projeto antes da fase de integração do CI [Deschamps et al. 2006]. Os FPGAs permitem o uso de linguagens de descrição de hardware *(HDL - Hardware Description Language)*, como *Verilog* e *VHDL*, para criação dos circuitos e as ligações entre eles, além das linguagens de verificação de funcionamento como *SystemC* e *SystemVerilog*.

Neste contexto, este trabalho apresenta o processo de desenvolvimento do protótipo de um jogo de corrida de carros, que opera em modo gráfico. Nele o jogador poderá controlar o carro utilizando botões para alterar a velocidade e visualizar o jogo em um monitor de padrão VGA. O cenário é composto por uma pista com obstáculos, como outros carros, simulando um ambiente de corrida. Na tela são exibidas informações sobre a distância percorrida e as "vidas" que restam ao jogador. Este modelo foi desenvolvido em linguagem Verilog e sintetizado em um dispositivo FPGA.

A abordagem utilizada neste trabalho envolve a descrição de todas as etapas de desenvolvimento, da concepção inicial até a etapa atual do projeto. Apresenta-se ainda uma breve descrição dos conceitos necessários para o desenvolvimento dos circuitos. Diante disso, serão mostrados os principais módulos do circuito, destacando suas funcionalidades. Por fim serão relatados os resultados apresentados até a presente etapa de desenvolvimento do projeto, bem como as perspectivas futuras.

2 Fundamentação

2.1 Visão Geral de um Dispositivo FPGA

O FPGA é um dispositivo lógico composto por um arranjo bidimensional de células lógicas e chaves programáveis genéricas. Nesta configuração, uma célula lógica pode ser configurada (programada) para executar uma função e as chaves são utilizadas para realizar a conexão entre as células [Chu 2008]. Essa configuração permite ao programador desenvolver sua lógica para obter funções a partir de uma ferramenta de *design*, a gravação de seu programa em um dispositivo FPGA, obtendo um modelo de circuito modificável. Um diagrama simplificado da arquitetura interna de um dispositivo FPGA pode ser vista na Figura 1.

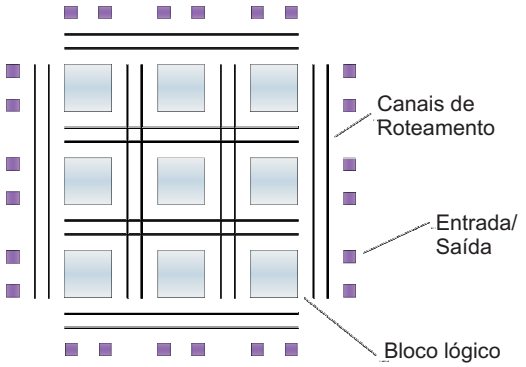


Figura 1: *Diagrama simplificado da arquitetura de um FPGA*

2.2 IP-Cores

Um *core* pode ser definido como a descrição de um sistema digital. Quanto à disponibilização, um *core* pode ser classificado como licenciável quando o fornecedor disponibiliza um projeto e um conjunto de ferramentas, conferindo a tarefa de projeto do sistema e sua fabricação ao cliente. Além disso um *core* pode ser definido como proprietário. Neste caso, a preocupação principal do fornecedor é proteger sua propriedade intelectual. Quanto à flexibilidade, um *core* pode ser classificado em três categorias: *hard*, *firm* e *soft*. A Tabela 1 mostra esta classificação [Palma et al. 2001].

A partir de uma análise da Tabela 1, pode-se classificar o *IP-core* descrito neste trabalho como do tipo *soft core*, uma vez que os sistemas se apresentam abertos a novas implementações e sua síntese pode ser transcrita para diversas tecnologias de lógica programável, sem alteração do seu funcionamento. Deste modo, a estrutura elaborada está apta a receber modificações de parâmetros de forma simples e independente.

2.3 Interface VGA

Em monitores do tipo CRT *(Cathode Ray Tube)* a imagem é formada pela emissão de elétrons através de um "canhão de elétrons"

Tabela 1: *Classificações de* IP-cores.

	Hard Core	***Firm Core***	***Soft Core***
Rigidez	A organização é predefinida.	Combinação de código fonte e netlist dependente de tecnologia.	Apresenta um código fonte comportamental independente da tecnologia.
Modelagem	Modelado como um elemento de biblioteca.	Combinação de blocos sintetizáveis fixos. Permite compartilhar recursos com outros cores.	Sintetizável com diversas tecnologias.
Flexibilidade	Não pode ser modificado pelo projetista.	A personalização de funções específicas é dependente da tecnologia.	O projeto pode ser modificado e independe da tecnologia.
Previsibilidade	Temporização é garantida.	Caminhos críticos com temporizações fixas.	A temporização não é garantida, podendo não atender os requisitos do projeto.
Proteção de Propriedade Intelectual	Alta. A descrição normalmente corresponde a um *layout*.	Média.	Muito pequena. Código fonte aberto.

sob uma superfície coberta por uma camada de material fosforescente, gerando um ponto na tela *(pixel)*. Em cada etapa é desenhado um novo ponto na tela até que se realize uma varredura completa na tela. A intensidade do brilho do ponto exibido é controlada pelo nível de voltagem introduzido pelo sinal de vídeo. O controle do caminho percorrido pelo feixe de elétrons é feito através de placas defletoras, horizontal e vertical, que produzem um campo magnético determinando o caminho que o elétron deve percorrer. Estas placas são responsáveis, portanto, pela identificação dos sinais de sincronismo.

Um controlador de vídeo é responsável por gerar os sinais de sincronismo e as saídas de dados para cada *pixel* de forma serial. O sincronismo horizontal (*hsync*) especifica o tempo requerido para percorrer uma linha, o vertical (*vsync*) controla o tempo necessário para percorrer toda a tela [Chu 2008]. Sendo assim, um circuito digital de controle dos sinais de sincronismos requer que sejam definidas, de forma objetiva, a disposição da parte visível da tela no monitor.

Monitores CRT, durante o sincronismo horizontal, apresentam uma pequena borda preta ao redor da imagem, definidas a partir dos valores de *back porch* (borda da esquerda) e *front porch* (borda da direita). Durante o sincronismo vertical também são verificadas regiões onde o sinal de vídeo deve ser desativado, regiões chamadas de *bottom border* e *top border* [Chu 2008]. Estes dados, além de todos os elementos que compõem uma varredura horizontal e vertical, são descritos na Figura 2. Nela é apresentado um diagrama dos ciclos de sincronismo horizontal e vertical para dispositivos CRT com resolução de 640x480 *pixels*, baseado no padrão VGA.

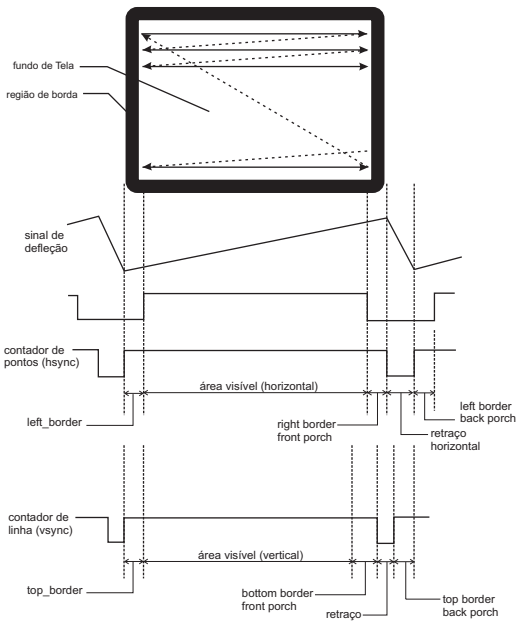


Figura 2: *Diagrama de tempo de uma varredura horizontal e vertical*

Analisando os sinais de sincronismo de um controlador VGA, existe ainda, além dos sinais de sincronismo *hsync* e *vsync*, o retraço horizontal e vertical. Assim como nas bordas, durante o retraço o sinal de vídeo deve ser desativado. No retraço horizontal, o sinal de vídeo deve ser desativado na região onde o feixe de elétrons retorna para a borda esquerda. No retraço vertical, o sinal deve ser desligado na região em que o feixe retorna para o topo da tela.

3 Desenvolvimento do Jogo

O desenvolvimento do projeto, ilustrado na Figura 3, baseou-se na criação do controlador VGA capaz de gerar sinais para realizar os sincronismos do vídeo através do uso de contadores. Este também serviu de suporte para o controle do posicionamento das imagens exibidas no monitor. O *core* é composto ainda pelo controle de jogo, responsável por responder aos sinais de entrada e processá-los, transformando-os em dados como velocidade e posição.

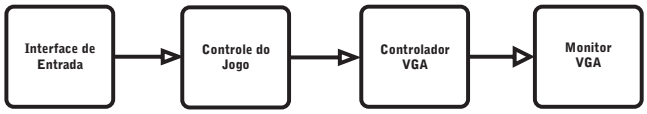


Figura 3: *Diagrama de blocos do projeto*

A implementação gráfica dos elementos do jogo é composta por *sprites* em duas dimensões, criados para representar imagens que se repetem, como carros e obstáculos. A seguir serão descritos os módulos desenvolvidos para o protótipo do jogo, incluindo os circuitos de controle, como o responsável por manipular os elementos na tela.

3.1 O Controlador VGA

O trabalho aqui descrito, teve origem em uma das avaliações do Estudo Integrado de Sistemas Digitais do curso de Engenharia de Computação. Este componente curricular engloba conceitos de Arquitetura de Computadores. Uma das restrições do problema era o uso do padrão VGA. Com a continuidade do projeto verificou-se que o kit FPGA disponível possui apenas 100 mil portas, um número pequeno para construir um *buffer* de memória. Devido a esta restrição, a interface gráfica do jogo é desenhada *pixel* a *pixel*, necessitando de pequenos blocos de memória para armazenar os *sprites*.

Basicamente, o controlador VGA é composto por contadores, a partir dos quais é controlada a exibição das imagens na tela. O diagrama do controlador está descrito na Figura 4. Estes sinais são utilizados para controlar o início e o final da área útil da tela, estabelecendo valores de *font porch* e *back porch*, bem como gerar os sinais de sincronismo vertical e horizontal necessários para o funcionamento correto do monitor de acordo com a resolução definida (640x480 *pixels*).

O projeto utilizou um sinal de *clock* de 32,768 MHz, definido pelo kit [Altera 2009], que estabelece a frequência máxima com que os

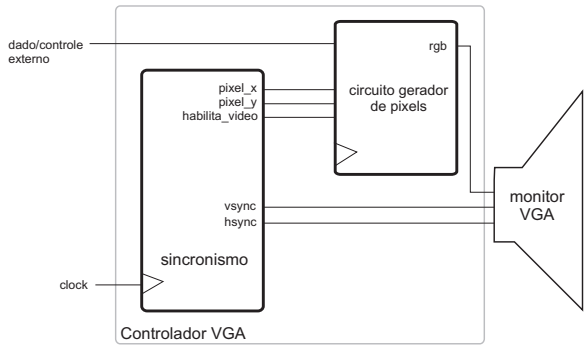


Figura 4: *Diagrama do controlador VGA*

pixels podem ser enviados ao monitor. Esta frequência é muito alta para os monitores disponíveis, que operam entre 60 Hz e 120 Hz, em sua maioria. Além disso, não é desenhado apenas um ponto na tela. Desde modo, foram projetados dois contadores, um para o sincronismo horizantal e outro para o vertical.

Ambos os contadores têm seus valores iniciados em zero. O contador horizontal é incrementado a cada transição de *clock* do FPGA. Inicialmente é enviado um pulso de sincronismo horizontal (*hsync*), indicando o começo de uma nova linha. Entre a faixa [0-96] do contador o vídeo é desabilitado, para garantir o processo de retraço. O contador vertical incrementa a cada ciclo horizontal completo, ou seja, quando uma linha de *pixels* é completada. Entre [0-2] do contador o vídeo é desativado, para garantir o retraço vertical. Neste intervalo é enviado um pulso de sincronismo vertical *(vsync)*, representando um ciclo completo de tela.

3.2 Controlador do Jogo

Inicialmente, foi necessário definir a área do jogo dividindo a tela em regiões onde serão desenhados o carro, a pista, as faixas laterais e centrais, que serão utilizadas para prover a noção de movimento e velocidade. Cada região representa uma condição para a análise da cor a ser exibida. A Figura 5 mostra as regiões e as cores escolhidas para as mesmas.

Figura 5: *Área do Jogo*

Os valores dos contadores vertical e horizontal são utilizados para representar as coordenadas (x,y) do *pixel* que será impresso. Para exibir imagens na tela, foram feitas comparações com os contadores horizontais e verticais. Quando os valores (x,y) estão dentro de uma das regiões definidas na Figura 5 o módulo comparador identifica a posição e envia a cor relativa à posição do *pixel* que está sendo impresso na tela, desse modo, são exibidas as figuras que compõem os elementos gráficos do jogo.

Os elementos gráficos que apresentam movimento dentro da imagem do jogo foram concebidos como *sprites*, imagens que são armazenadas em matrizes na memória do FPGA. Cada matriz armazena valores binários, sendo que cada bit indica um *pixel*, e os bits que possuem valor igual a 1 representam *pixels* que serão exibidos na tela. Alguns *sprites* são compostos por várias cores, para isso, eles são necessariamente desenvolvidos em matrizes separadas, cada uma relacionada à uma, quando a imagem é exibida na tela mostra-se a figura composta por todas as matrizes.

Para a exibição de cada *sprite* é necessário definir parâmetros relativos à sua posição. O carro foi definido em duas matrizes de 60 linhas e 120 colunas, totalizando uma região de 720 *pixels*. A primeira matriz armazena o desenho da carenagem do carro, e a segunda guarda os desenhos das rodas e capacete do piloto.

3.2.1 Movimentação do Carro

O efeito de movimento foi desenvolvido alternando as posições das figuras. Foram criados retângulos que representam faixas na lateral e no centro da pista, e a cada um deles foram associadas variáveis contendo os vértices. O movimento é obtido pelo incremento desses vértices de acordo com a velocidade atual, deste modo, a figura é exibida em uma nova posição, até chegar ao limite da tela. Quando o limite é alcançado, o parâmetro recebe o valor inicial, e recomeça o ciclo. A velocidade é estabelecida de acordo com o valor de entrada do botão de velocidade, sendo que, a cada vez que o botão é acionado, a variável de velocidade é incrementada de um valor previamente definido. A Figura 6 mostra o diagrama do controlador do jogo.

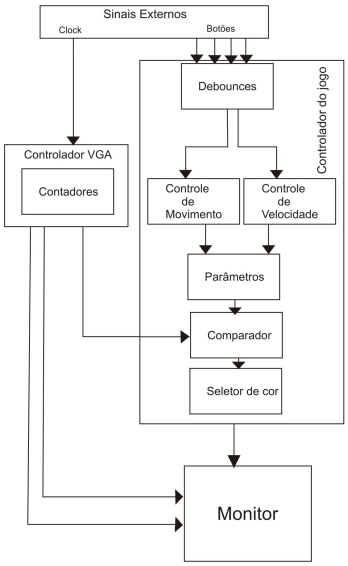


Figura 6: *Detalhe do controlador do jogo*

Inicialmente, o protótipo utiliza botões como interface com o usuário para a movimentação do carro. Para evitar a ação de ruídos e de repetições quando um botão se mantiver prescionado foi necessária a implementação de *debounces*, que geram sinais estáveis para o módulo do jogo. O circuito do *debouncer* foi desenvolvido como um módulo e embarcado no dispositivo FPGA. A Figura 7 ilustra a ligação dos botões de entrada.

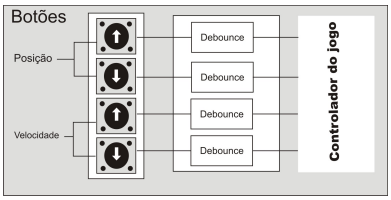


Figura 7: *Diagrama da interface de entrada*

O carro apresenta uma posição relativa chamada de âncora de *sprite*, que possibilita a movimentação da imagem na tela. Para a movimentação do carro é necessário alterar o valor da âncora, que é modificado realizando-se operações de incremento e decremento, de acordo com o acionamento dos botões. Ao somar o valor da posição inicial, consequentemente o carro é movido para uma posição inferior na tela. Enquanto que, subtraindo valores, o carro é movido na direção da porção superior. Para impedir que o carro

saia da pista, foram definidos parâmetros limitadores de posição, que indicam os limites verticais e horizontais para deslocamento do carro.

3.3 Resultados

A ferramenta utilizada para o desenvolvimento dos circuitos lógicos e o estabelecimento das ligações foi a *Altera Quartus II 9.0 Web Edition* e o dispositivo programado foi o *FPGA Starter Kit Altera (Pl1K100A)* [Altera 2009]. A interface de entrada e saída do protótipo foi implementada em uma matriz de contatos, onde foi desenvolvido o circuito que proporciona a interface VGA para a placa FPGA. O padrão VGA utiliza conector D-Sub de quinze pinos, sendo que nem todos são utilizados, apenas os sinais de sincronismo e do RGB.

A matriz de contatos possui também quatro botões do tipo *push-button*, dois deles são utilizados na movimentação lateral, e os demais para o controle de velocidade. O algoritmo do jogo fica armazenado na memória interna do FPGA, onde são executadas todas as rotinas necessárias para a execução do algoritmo implementado. A Figura 8 mostra o circuito montado para a interface do jogo.

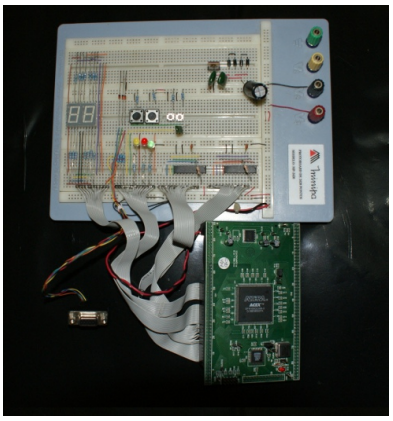


Figura 8: *Circuito de interface E/S*

O teste inicial do protótipo foi realizado com uma imagem estática da tela do jogo. Em seguida foi adicionado o *sprite* do carro fixo e as faixas foram colocadas em movimento. Depois dessa etapa de teste, foi implementada a interface de entrada testando se havia incremento de posição no carro e da velocidade de passagem das faixas. Por fim, foram testados os limites de posição e velocidade, executando o jogo e acionando os botões para verificar se os limites eram respeitados. A Figura 9 mostra uma foto retirada de um dos testes de funcionamento do protótipo.

Figura 9: *Teste do protótipo*

4 Conclusão

Este trabalho apresentou a etapa inicial de desenvolvimento de um jogo de carros em FPGA, demonstrando as características dos circuitos desenvolvidos, além da interação dos mesmos para compor o sistema. Para a exibição das imagens na tela foi implementado um módulo controlador VGA, este gerencia todos os valores de parâmetros que dinamizam a disposição das imagens na tela.

O resultado parcial do projeto compreende o desenvolvimento da interface gráfica do jogo, do controle de velocidade e deslocamento do carro. Através dos sinais gerados pelos botões, a movimentação do carro e variação da velocidade são controladas. Ao pressionar o botão de movimento, o carro desloca-se na direção vertical, para possibilitar o desvio dos obstáculos.

A próxima fase do projeto prevê o desenvolvimento do controle de colisões do carro, a contagem da quilometragem percorrida, o gerenciamento das vidas e a colocação de controles analógicos para a velocidade e deslocamento do carro. Com o intuito de melhorar a aparência e aumentar a sensação de movimento do jogo, pretende-se inserir algumas figuras na grama (área verde da tela). A Figura 10 é um exemplo de como será uma tela da versão final deste jogo.

Figura 10: *Perspectivas futuras*

Referências

ALTERA, 2009. Fpga, cpld and asic from altera. `http://www.altera.com`.

CALAZANS, N. L. V. 1998. *Projeto Lógico Automatizado de Sistemas Digitais Sequenciais*. Master's thesis, Escola de Computação, Universidade Federal do Rio de Janeiro.

CHU, P. P. 2008. *FPGA Prototyping by Verilog Examples*. WILEY.

DESCHAMPS, J.-P., BIOUL, G. J. A., AND SUTTER, G. D. 2006. *Synthesis of Arithmetic Circuits - FPGA, ASIC, and Embedded Systems*. John Wiley and Sons, Inc.

PALMA, J. C., MORAES, F., AND CALAZANS, N. 2001. *Métodos para Desenvolvimento e Distribuição de IP Cores*. Master's thesis, Faculdade de Informática - PUCRS.

SCHALLER, R. R. 1997. Moore´s law: Past, present and future. *IEEE Spectrum 34*, 6 (June), 52–59.

Development of an interactive game using a webcam

Edmar Souza Jr. Gabriel A. Delgado Rudimar L. S. Dazzi Natália Ellery

Grupo de Inteligência Aplicada – Centro de Ciências Tecnológicas da Terra e do Mar – Universidade do Vale do Itajaí (UNIVALI)
Rua Uruguai, 458 - 88302-202 – Itajaí– SC – Brasil

Abstract

The electronic games are becoming each time more popular and using resources each time more common in the personal computers like the webcam. Actually some games allow to be played using just the webcam and the body motion. This game category is known as webcam games. This project has as objective research the motion capture technology used by the webcam games. The motivation for this project was the possibility of use this resource to create educational games and games focused in physical rehabilitation on a dynamic and funny way. Thus it was necessary to develop research on existing motion capture technologies used in the current games. Adobe Flash was chosen as development platform, using the AS3 (Action Script 3) programming language. As result, was created a game named "Coleta Seletiva" that allows to identify in a clear way, its behavior and future uses of the researched technology.

Keywords: webcam, motion capture, Adobe Flash

Authors' contact:
{edmar.souza, rudimar}@univali.br
gabrieldelgado@yahoo.com
nataliaellery@gmail.com

1. Introduction

The world of electronic games grows continuously. The games alternatives and realism are impressive, even for who doesn't follow the genre. Until a little time ago the unique dispositive used in games was the joystick, which allows the control of the whole game. Following the personal computer's popularization, besides the joystick, the games started to use the mouse and the keyboard allowing the gamers to control the game with the computer's basic resources.

However, these resources are little ergonomics and uncomfortable when used for long periods of time. They can even cause Repetitive Strain Injury Syndrome (RSI).

Recently, Nintendo launched its new console, the Nintendo Wii, assuming the leadership of the new generation games, bringing to Nintendo its "original" position after 17 years [UOL JOGOS, 2007]. Nintendo Wii has a joystick called Wii remote (simply Wiimote), this device is an motion capture controller, that capture the movements made by the player while he's manipulating the controller, like an "air-mouse". [WIKIPEDIA, 2008]

Following this evolutional tendency one of the alternatives found by the game developers to make them more interactive, is the motion capture using a webcam.

Games using this technology are emerging today in the market allowing the player to control the game moving their own body according to the needs of the game. Two significant examples are the games Eyekanoid [POINT OF THE GAME, 2008] and Playdojam [PORTALCAB.COM, 2008]. In the first, the player controls the bar to hit the ball using hands movement, and it follows the movement done by the player's hand. In the second game, the player's image is displayed in the center of the screen, inside a basketball game, and the player becomes part of the game, rebutting or throwing the ball towards the basket.

A research on the motion capture technology using webcam was started, technology that had not yet been investigated in the UNIVALI's computing courses. The objective was to develop a platform to develop games based on this technology, generating as example, games using this behavior.

Initially the main focus was to acquire knowledgement on the area and to produce material for future projects. This way, the game "Coleta Seletiva" was developed, which demonstrates the operation of the technique and the interaction of the player with the game through a library that was named Barbara who uses the webcam.

The game "Coleta Seletiva", developed using the Barbara library, is an educational game aimed to children aged 4 to 6 years.

2. Related Work

A pioneering work on interaction in real time with images from the cameras is Artificial Reality of Myron W. Krueger, 1969. In this work [KRUEGER, 1991] are described and presented studies and developments on innovative ways to interact with computers through the use of human movement without the need for any hardware attached to the person. Such movements were captured by a video camera and used as input for the system.

In Videoplace [KRUEGER, 2006], Krueger has a system to allow a person to interact with virtual objects presented through graphs generated by a computer.

Krueger was able to demonstrate the potential of more than 50 different modes of interaction through his tests, through the construction of prototypes in which the person interacted with virtual objects in some way, for example, drawing pictures with your fingertips, playing in a virtual environment with a bubble, trying to follow dance moves shown on a monitor or interact with virtual balloons that float freely.

In the famous Kids Room [BOBICK, 2000], held at MIT (Massachusetts Institute of Technology), a room was developed in order to attract the attention of children and make them interact in a closed environment.

This is a children's room that tells stories through images projected on walls, lights and sound environment to increase the immersion of children in history. This study used computational vision to promote interaction with virtual objects and to control the intensity of physical effort required by children who are in the room, depending on how active they are.

This new interaction form enabled the development of a new category of computer games named "webcam games" in addition to the conventional model of interaction formed by joystick, keyboard and mouse. According to Paula, Miranda Neto and Bonini (2008), the conventional mode of interaction promotes poor posture and inactivity, in which little physical activity is carried out during the game than to cause repetitive strain injuries, or simply, as quoted by [Abraham and Nath, 2004], making uncomfortable to use for a long period of time.

So, the use of natural movements of the hand or body to control actions in a game provides a more comfortable and easy way to play (Abraham and Nath, 2004).

These and other scientific and technological evolutions enabled the concept of Augmented Reality (AR) that brings the games to the computer's virtual space of the player.

AR is the overlay of virtual objects generated by computer in a real environment, using it for some technological device [Kirn and Tori 2004 apud Zorzal ET AL 2006]. This definition of AR is part of a broader context called Mixed Reality (MR).

The concept of MR concerns the actual existence of elements belonging to the world in which users live and synthetic, created by computer, in the same environment. Depending on the way in which this combination occurs, the degree of reality and virtuality, applications are classified into different sub-areas, including Virtual Reality (VR) (also called Increased Virtuality) and AR. While in VR is a dominance by virtual objects, where an entire world is modeled by computer in AR which is the very real world is mixed with some synthetic artifacts, added in real time, in a way that they seems to belong to the environment that they are added [Milgram and Kishin, 1994 apud LIMA, 2007].

In other words, MR is the combination of real environment with the virtual environment generated by computer. It can receive two denominations: Increased Reality when the environment is the real principal, and Increased Virtuality, where the virtual environment is the main environment. Thus, the AR is a particularization of mixed reality [Zorzal et al, 2006].

Actually, AR is used in various fields of application, such as entertainment, medicine, education and maintenance (LIMA, 2007). Further on making virtual objects can be placed in real environments, the AR also allows the user to interact with virtual elements using their hands, thus eliminating complex technological devices and making the mixed interaction with the environment much more enjoyable, attractive and motivating [Santin et. al. 2004 Zorzal et al 2006].

Several programming languages have been used to create prototypes such as Java [Abraham and Nath, 2004] and the C language [Paula, Miranda Neto and Bonini, 2008] and [Zorzal et al 2006]. But in all implementations remain the need for a webcam to capture the movements of the player. These images are analyzed by a computer, processed and displayed on a monitor. There is the possibility of projecting the image generated in a wall, facilitating for the players visualization of the elements which they can interact.

Here are some webcam games examples:

- Eyekanoid (Figure 1);
- Playdojam (Figure 2);
- Flights Over Sahara (Figure 3);

Figure 1: Eyekanoid (http://cam.playdo.com/play_eye.htm)

Figure 2: Playdojam. (http://www.playdojam.com/)

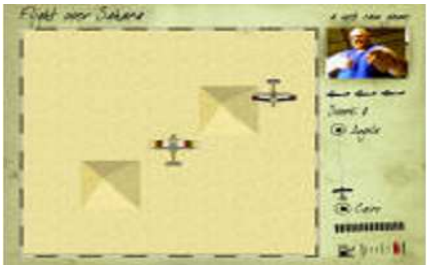


Figure 3: Flights over Sahara (http://www.motiongames.net/)

3. Development

In the game Coleta Seletiva, the child / player aim to not let the trash fall out of the game screen. The aim of the game with the observation of children in this age group, is that they knows how to interact with the tool, gathering all the falling garbage with a trash, in a coordinated manner. The game is ready to react in different ways to the garbage collected or not, but it is not necessary that the child win the game but that is in constant movement to collect the garbage, earning points.

This is a playful game that seeks only to identify the characteristic features of the garbage. The movement required is small and only horizontal, to facilitate the gameplay, since the age in question (from 4 years old) haven't the total development of motor coordination.

The game developed following the idea of games like Playdojam or Eyekanoid, but the "body" of a educational game. So the game was contextualized in the process of collection, causing the child to identify what is garbage and collecting them.

Figure 4 shows a game screenshot, where you can see the elements falling and the trash where they should be collected. The handling of the garbage is made by moving a yellow object, held in the player's hand (or a glove). The trash will follow the yellow object caught by the webcam, even if this object is the player's shirt (in which case the player needs to move from one side to the other to move the garbage).

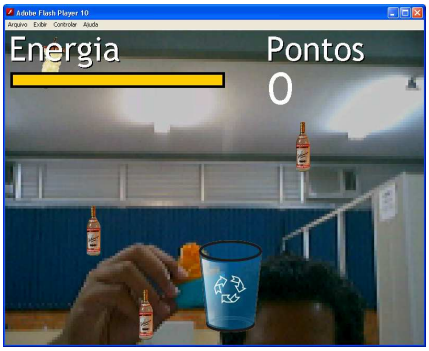


Figure 4: The game Coleta Seletiva

The choice of a yellow object to provide a reference was made to facilitate handling, as the children can more easily relate a real object with the game trash than with the own hand movement. The yellow color was chosen because it is a living color and easy to find on various objects.

There was been developed a research on the required technology to develop interactive games using a webcam. With the results of the research, the Barbara library was developed to make the object motion and color capture easier. Barbara was written in Action Script 3 programming language, using the tool Adobe Flash.

One way to motion capture by color, implemented in the Barbara library works in as follows:

- The program capture the webcam's current image
- The image is scanned, looking for items that match the color set by user.
- When an object is found, a rectangle is made using the edges of the points where the object was found.

Figure 5 shows an example of how such behavior is detected in tests performed during the project.

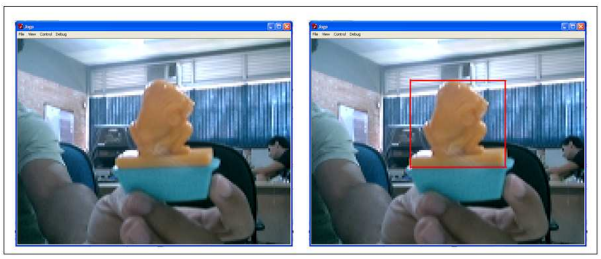


Figure 5: Object detection using colors.

4. Tests and results

Trials were made out with three different objects for the trash movement, one of them, a yellow glove, another is a glass that looks like the game trash itself, and the third is a shirt that a child was wearing. The children's reactions were different in all tests, and the glove's use was the most practical, because allowed movement freedom to children play the game, but the glass has a very interesting effect, because the children would actually pick up trash with the glass, simulating a real collection of garbage.

The use of the shirt was a different experience, because the children needed to move from side to side to collect the garbage that fell, it created an hyper active effect in some children, making them start running from one side to the other in a frenzy. This fact make the meaning of the game get lost, making the children have fun because of the moving part, and not because of the game meaning itself.

Figure 6 illustrates two pictures of the game play, during the tests made in the Colégio de Aplicação Univali, with children aged from 4 to 6 years old. In tests it's always utilized an environment with computer, webcam and multimedia projector, so the children could follow the game of the other colleague in the projection. A child that's playing interacts with the image of the monitor.

Figura 7. Tests with children of the Colégio de Aplicação Univali. - CAU

With the tests we can realize that the children interacted very well with the game, and enjoyed it even more. The fun is a part of the game and also a part of the interaction fun. Another important point is the immersion that the game created to these children and the degree of motivation generated.

Besides these aspects, the fact that there's no need of a joystick or other device, with buttons and options, made the children play more naturally, even a digital game.

5. Conclusions and Future Work

Children aged between 4 and 5 years old, sometimes, are excluded from some games, because they can't read yet, and also don't know how works input devices such as mouse, keyboard and joystick. The lack of motor coordination is a problem for children to reach the end of the game, and get unmotivated to continue playing, besides that, there is always a concern with the tool utilized to play the game, because children could drop it into the floor or break it.

The Computer Vision can help to solve this problem, making these children, to participate of the digital world and to have access to electronic entertainment.

There are several applications of this interface, guided by visual signals, to be applied in games: control of the mouse and recognition of gestures and postures. The interfaces to these features may be more intuitive to the present time and promote access for people with special needs. With this purpose, the library development has shown to be very versatile and of easy application for the development of the applicative described in this project.

Besides the library, the software developed has been tested by children of CAU Airplane Group - College Application UNIVALI, showing that it may be an ally to the development of children's motor coordination in the range of 4 years old. The children's motivation in the game was a strong point in the tests, but it still needs some adjustments in the speed of movement, size and shape of objects, fulfill needs of that children's age.

The explanation of this fact makes reference to the phrase: "general specific motor coordination" that's the kind of coordination that allows a child or adult to control the body in space, controlling the movements more harsh. Ex: Walk, Jump, crawl [GALLAHUE, 2004]. It is well developed through the Coleta Seletiva game, because the distance determined to make sure that children are well positioned with the object in webcam, makes them have to give 4 steps to be able to get of one side to another on the screen.

The tool is working as proposed: it collects the garbage, sum score, control the energy bar, the use of a yellow object in the capture of the webcam image, has reached the goal of moving the trash, and garbage randomly fall as planned.

Due to the test realized with the CAU's children, we could observe that some adjustments are necessary. Examples: The speed that the garbage falls should be reduced; the sounds emitted shall only be positive and motivating; the score, because children can't count yet, should be run with visual images or sounds; the sum and subtraction figures which were to improve energy bar, had no response differentiated from the others, showing that child that age don't have actually a need to win, but when they lose the game, they got unmotivated.

References

ABRAHAM, Ajay; NATH, Nitendra. Computer Vision For Computer Games. 2004. Disponível em: <http://www.ces.clemson.edu/~stb/ece847/fall2004/projects/proj19.doc>. Acesso em: 08 mar 2008.

BOBICK, AARON F. INTILLE, STEPHEN S. DAVIS, JAMES W. BAIRD, FREEDOM.

GALLAHUE, D. L., & Ozmun, J. C. (2006). Understanding motor development: Infants, children, adolescents, adults (6th ed.). Boston: McGraw-Hill (5th ed., 2002; 4th ed., 1998; 3rd ed., 1995; 2nd ed., 1989; 1st ed., 1982, Portugese Editions, 2001; 2005.)

KRUEGER, MYRON W. Artificial Reality II. Addison-Wesley. 1991.

KRUEGER, MYRON W. Videoplace [online]. Disponível em: http://www.jtnimoy.net/itp/newmediahistory/videoplace/. [acesso em: 10 dez. 2006].

LIMA, João Paulo Silva do Monte. Um framework de Realidade Aumentada Para o Desenvolvimento de Aplicações Portátesi para Plataforma Pocket PC. Trabalho de Graduação em Ciência da Computação - UFPE. 2007.

PAULA, Luis Roberto Pereira de; BONINI NETO, Renato; MIRANDA, Fábio R. de.Câmera kombat - interação livre para jogos. Disponível em: <http://cin.ufpe.br/~sbgames/proceedings/files/camera%20kombat.pdf >. Acesso em: 07 mar. 2008.

PINHANEZ, CLAUDIO S. CAMPBELL, LEE W. IVANOV, YURI A. SCHÜTTE, ARJAN AND

POINT DO GAME. Eyekanoid, Disponível em: <http://pointdogame.blogspot.com/2007/11/web-cam-games-eyekanoid.html >. Acesso em: 10 mar. 2008.

PORTALCAB.COM. Playdojam, Disponível em: <http://www.portalcab.com/joguinhos/playdojam.php>. Acesso em: 10 mar. 2008.

UOL. Wii chega à liderança do mercado de videogames. Disponível em: http://jogos.uol.com.br/wii/ultnot/2007/09/12/ult4097u955. jhtm >. Acesso em: 08 mar. 2008.

WILSON ANDREW. The Kidsroom. Communications of the ACM. 2000. 43(3), 60-61.

WIKIPEDIA. Nintendo Wii, Disponível em: < http://pt.wikipedia.org/wiki/Wii>. Acesso em: 07 mar. 2008.

ZORZAL, Ezequiel Roberto; CARDOSO, Alexandre; KIRNER, Claudio; LAMOUNIER JUNIOR e Edgard, 2006 Realidade Aumentada Aplicada em Jogos Educacionais.

Elegy: Aplicando o Processo de Fábrica de Jogos ao Domínio Role-Playing Games

Leila S. de S. T. de Azevedo André L. de M. Santos André W. B. Furtado

Universidade Federal de Pernambuco, Centro de Informática, Brasil.

Resumo

Este paper apresenta uma solução de fábrica de jogos, denominada Elegy, para o domínio de jogos RPG, em busca de obter um maior custo/benefício durante a produção desses *softwares*. Para a elaboração da Elegy foram utilizados conceitos como *domain-specific languages* (DSLs) e uma metodologia de análise de domínio para jogos intitulada SharpLudus.

Palavras-chave: DSL, Role-Playing Games, Fábrica de Jogos.

Authors' contact:
{lssta,awbf,alms}@cin.ufpe.br

1. Introdução

A indústria de jogos eletrônicos é uma das mais expressivas no ramo do entretenimento [1], equiparado-se às indústrias de cinema e música. Porém, a indústria de software em geral ainda hoje funciona de forma artesanal, desta forma a produção não atende de forma eficiente à demanda de mercado[2].

Neste cenário, surgem opções para automatizar a produção de software. Uma abordagem interessante é a implementação de fábricas de software, baseada na produção de softwares de uma mesma família, através do uso de linguagens, *frameworks* e ferramentas [2]. Neste paradigma são montadas verdadeiras linhas de produção de software, focadas em soluções para problemas de uma mesma espécie.

Alinhado à iniciativa de Fábricas de Software, encontramos o conceito de Desenvolvimento de Domínio Específico, abordagem utilizada para resolver problemas que pode ser aplicada quando um problema em particular ocorre repetidamente [3]. Baseia-se no conceito de Linguagens de Domínio Específico (*Domain-Specific Languages* – DSLs), forma limitada de linguagens computacionais projetadas para uma classe específica de problemas que oferecem, através de notações e abstrações apropriadas, expressivo poder de foco, e usualmente restrito, a um problema do domínio.

Este trabalho apresenta uma alternativa, baseada no uso de Fábricas de Software e Desenvolvimento de Domínio Específico, para produção de jogos no domínio *Role-Playing Games*.

2. Análise de Domínio

Por quase três décadas, a área de Análise de Domínio (Domain Analysis ou DA) tem servido outras iniciativas da Engenharia de Software, como Desenvolvimento baseado em Componentes, Linhas de Produção de Software e Reuso de Software em geral [4]. Segundo Prieto-Diaz [5], DA é definida como um processo pelo qual a informação utilizada no desenvolvimento de sistemas de software é identificada, capturada e organizada com o propósito de fazê-la reusável na criação de novos sistemas.

O termo Análise de Domínio foi inicialmente introduzido por Neighbors [6] como "a atividade de identificar os objetos e operações de uma classe de sistemas similares num domínio particular de problemas". Ainda segundo ele, "a chave para softwares reusáveis é capturada na análise de domínio na medida em que se salienta a reusabilidade da análise e *design*, não código".

Porém, nem Neighbors nem Prieto-Diaz, dentre outros autores, resolveram o problema de como fazer a análise de domínio. É necessário desenvolver abordagens para análise de domínio, a fim de explorar verdadeiramente o reuso em ambientes industriais.

A metodologia SharpLudus proposta por Furtado [4], propõe uma abordagem pragmática de análise de domínio para a criação de fábricas de jogos, que engloba as atividades de definição de características sob análise, seleção de amostras da linha de produção, execução da análise e, finalmente, a definição das características comuns e variabilidades.

Esta metodologia foi considerada mais adequada por ser focada na automatização no desenvolvimento de jogos, e portanto escolhida para ser aplicada na criação da fábrica *Elegy*. Nas próximas seções encontra-se o resumo de cada etapa do processo no domínio RPG.

2.1 Funcionalidades do domínio sob Análise

Nesta etapa foram definidas funcionalidades e características consideradas importantes no domínio de jogos RPG eletrônicos. A Tabela 1 apresenta as variáveis escolhidas para análise.

Tabela 1: Variáveis de Análise.

Funcionalidade	Descrição
Descrição	Principais características do jogo em geral.
História	Breve descrição do enredo.
Fluxo da História	Descrição de como se dá o desenrolar da história, linearidade, presença ou não de missões alternativas e impacto das ações dos jogadores na história.
Personagens	Características, atributos e habilidades de personagens jogáveis, presença ou não de personagens não jogáveis, quantidade de personagens que podem ser controlados.
Desenvolvimento dos Personagens	Como os personagens evoluem no decorrer da história.
Cenário	Descrição do mundo e época em que se passa a história.
Sistema de Batalha	Descreve como as características inerentes aos personagens podem afetar no resultado das ações de batalha. Lógica e visão estratégica da batalha também são pontos cruciais neste item.
Fluxo em Batalha	Ações que podem ser tomadas dentro da batalha e como os personagens podem interagir neste ambiente.
Fluxo fora de batalha	Ações que podem ser tomadas fora da batalha e como os personagens podem interagir.
Itens	Itens e equipamentos que podem ser encontrados durante o jogo.

2.2 Seleção de Amostras

Foram escolhidos 5 jogos considerados representativos no domínio:

- Final Fantasy VI
- Chrono Trigger
- Baldur's Gate II
- Seiken Densetsu III
- Neverwinter Nights

2.3 Execução da Análise de Domínio

Nesta etapa as variáveis escolhidas anteriormente foram analisadas em relação a cada um dos jogos escolhidos.

2.4 Extração de Semelhanças e Variabilidades

Nesta etapa as funcionalidades semelhantes e variáveis foram organizadas através de *feature models*. Para tal, foi utilizada a ferramenta Feature Model DSL [7]. A Figura 1 mostra a raiz dos diagramas.

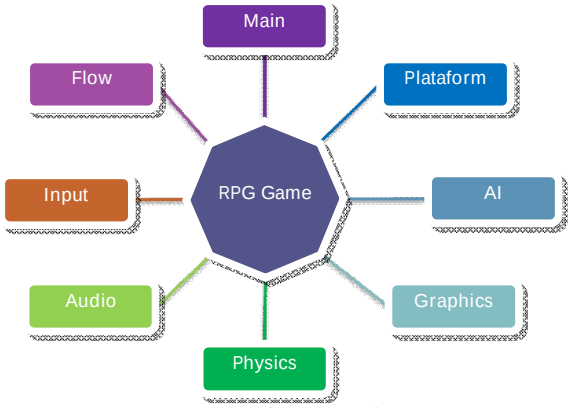


Figura 1: Variáveis de Análise.

2.5 Validação do Domínio

Nesta etapa, com o *feature model* pronto, uma amostra que não foi incluída na análise será usada para determinar a generalidade do modelo, ou seja, o jogo escolhido deverá ter suas funcionalidades mapeadas no *feature model* projetado. O domínio foi validado pelo jogo *Persona 3*, da série *Megami Tensei*, lançado em 2006 pela *Atlus* para a plataforma *Playstation 2*.

2.6 Critério de Parada para a DA

O tempo foi o critério levado mais fortemente em conta para a finalização da análise. Como os jogos RPG costumam ser bastante complexos, o número de funcionalidades mapeadas no modelo certamente representa apenas um subconjunto, ainda que bastante representativo e generalizado. Desta forma termina o processo de análise de domínio e funcionalidades, passando agora para o momento de implementação da fábrica.

3. Elegy Game Factory

O resultado obtido da análise de domínio, no sentido *top down*, e o estudo da *engine RPG Starter Kit* [8], escolhido como *domain framework* (a ser consumido pelo código gerado por DSLs da fábrica), foram utilizados para guiar a modelagem, através da combinação de abordagens top-down e bottom-up.

A abordagem top-down abrange toda aplicação da metodologia sobre o domínio, resultando nos feature models, ao mesmo tempo em que é feito um estudo em cima da engine a ser reusada, no sentido bottom-up. O resultado da aplicação destas duas abordagens em paralelo culmina na modelagem da linguagem visual, juntamente com o código a ser gerado. Desta forma uma camada de abstração surge de modo a facilitar a criação de jogos através do uso de elementos do domínio, como personagens, mapas, inimigos, quests, entre outros, sem a necessidade de programação em linhas de código.

Foram modeladas 3 DSLs para compor a fábrica: Map Manager DSL, Quest Manager DSL e Entity Definition DSL. Estas DSLs foram modeladas separadamente, mas como pontos de integração entre si. O escopo deste projeto não abrange a integração entre as DSLs.

Nas próximas seções serão apresentadas as DSLs modeladas no *Microsoft DSL Tools* [9]. Cada DSL gera código XML para a plataforma XNA. Este código XML é consumido pela *engine* RPG Starter Kit para a criação de jogos.

A grande contribuição deste projeto esta justamente na geração do código XML a ser consumido, pois esses arquivos muitas vezes são extensos e complexos para serem escritos manualmente, tornando desta forma a criação de jogos enfadonha e lenta.

3.1 MapManagerDSL

Esta DSL é responsável por gerenciar a criação de mapas e a transição entre eles.

Através desta DSL podem-se criar diversos mapas, com transições entre eles, como mostra a Figura 3. Para definição de *tiles* e objetos do mapa usa-se um editor, como pode ser visto na Figura 2. Este editor também exibe opções como definição de pontos de colisão no mapa e por onde se dará a transição de um mapa para outro.

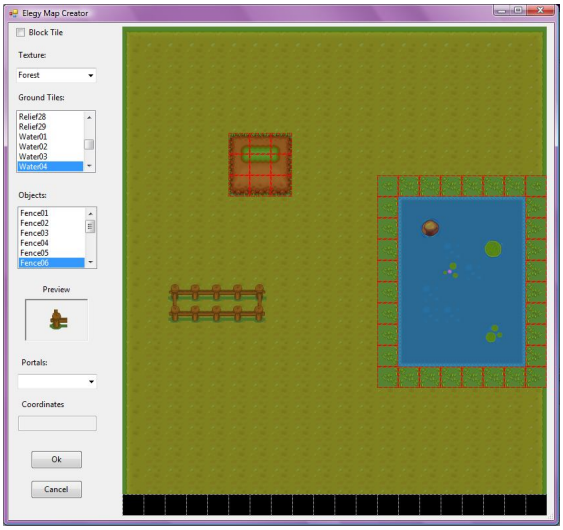


Figura 2: Editor customizado de propriedades do mapa.

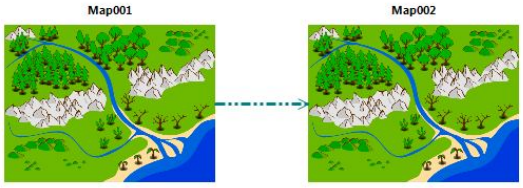


Figura 3: Representação visual de mapas e transições.

3.2 QuestManagerDSL

Esta DSL é responsável por gerenciar a criação de *quests*.

Através desta DSL pode-se definir uma *quest*, como pode ser visto na Figura 4.

Figura 4: Representação visual de uma *quest*.

Uma *quest* apresenta características como itens e inimigos requeridos, recompensas, entre outros, que são modificados através de editores customizados.

3.3 EntityDefinitionDSL

Esta DSL é responsável pela criação de personagens, tanto personagem principal, inimigos e NPCs de quests.

O personagem principal, inimigos e NPC's podem ser definidos como mostra a Figura 5.

Figura 5: Representação visual dos personagens.

Editores para modificar as propriedades dos personagens também foram implementados, como o editor de escolha de *sprites*, representado na Figura 6.

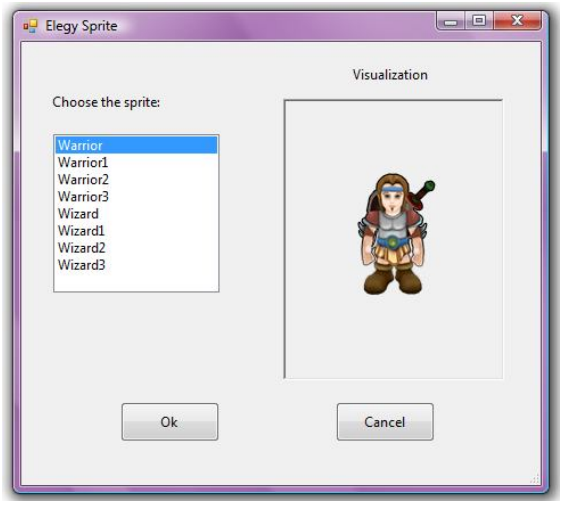


Figura 6: Editor de escolha dos sprites do personagem.

4. Conclusão

A importância deste trabalho não se resume à visualização de uma ferramenta consolidada e com grande potencial. O grande desafio na implementação produtiva de *games* através da abordagem de Fábricas de Software está na escolha de um processo adequado, confiável e que traga resultados satisfatórios. Este processo deve abranger todas as etapas de elaboração do projeto, desde sua idealização, design, até sua implementação.

O alvo do projeto Elegy esteve justamente no estudo e aplicação da metodologia SharpLudus durante a etapa de análise de domínio e funcionalidades, trazendo contribuições para esta metodologia, que ainda está em produção.

Para validar o projeto de DSL é necessário um framework, que consumirá o código gerado. A *engine RPG Starter Kit* foi escolhida para esta finalidade, com a vantagem de trazer portabilidade para a plataforma XNA. Desta forma a criação da Elegy não foi feita de forma linear. Enquanto a análise era processada, um estudo sobre a engine foi feito e culminaram na produção das três DSLs apresentadas na seção 3.

Muitos desafios foram encontrados devido à engine não ter sido produzida propriamente para a Elegy. Alguns deles puderam ser contornados, fazendo com que a criação de jogos RPG com esse motor de jogos se tornasse mais produtiva e interessante. Outros, como a necessidade de integração de DSLs, não puderam ser resolvidos nesta versão.

5. Trabalhos Relacionados

Este projeto teve como iniciativa a fábrica de jogos SharpLudus, proposta por Furtado [4]. A Elegy visou complementar a fábrica SharpLudus, que foi inicialmente focada em jogos do domínio *arcade*. Assim como a SharpLudus, a Elegy utiliza o conceito de análise de domínio e desevolvimento de domínio específico, com uso de DSLs para gerar o código a ser consumido pela *engine* escolhida.

Referências

[1]ESA. (2007). Essential Facts About the Computer Game Industry. Acesso em 1 de Março de 2009, disponível em www.theesa.com/facts/pdfs/ESA_EF_2008.pdf
.

[2]GREENFIELD, J., ET AL(2004). Software Factories: Assembling Applications with Patterns, Models, Frameworks and Tools. Wiley & Sons, 2004.

[3] FOWLER, M. Language Workbenches: The Killer-App for Domain Specific Languages? Acesso em 26 de Março de 2009, disponível em martinfowler.com/articles/languageWorkbench.html,;

[4]FURTADO, A. W. (2006). SharpLudus: Improving Game Development Experience through Software Factories and Domain-Specific Languages.Dissertação de Mestrado. Recife: UFPE.

[5]PRIETO-DIAZ, R. (1990). Domain Analysis: An Introduction. ACM SIGSOFT .

[6]NEIGHBORS, J. M. (1980). Software Construction Using Components Software Construction Using Components.

[7]FURTADO, A. W. (2008). Feature Model DSL. Acesso em 22 de Abril de 2009, disponível em http://www.codeplex.com/FeatureModelDSL
.

[8]RPG STARTER KIT. Acesso em 1 de Março de 2009, disponível em http://creators.xna.com/en-US/starterkit/roleplayinggame.

[9]MICROSOFT DSL TOOLS. Disponível em http://msdn.microsoft.com/en-us/library/bb126235.aspx.

Especificação do Comportamento de Agentes Virtuais Inteligentes por Demonstração

Gênesis B. Campos Vladimir O. Di Iorio Carlos R. Marques Jr. Alcione P. Oliveira José Luís Braga

Departamento de Informática – Universidade Federal de Viçosa – Brasil

Resumo

A possibilidade de testar hipóteses e experimentar a realidade sem as despesas e perigos do mundo real fez com que a utilização de ambientes virtuais atraísse a atenção de diversos profissionais. Tão importante quanto interagir em ambientes realistas é lidar com personagens que transmitam a sensação de vida, seres "inteligentes". A definição da camada de seleção de ações é determinante para o realismo do personagem e pode ser identificada a partir da exemplificação por parte do usuário do comportamento que deve ser executado pelo agente. Diante da não exploração de tal característica pelas abordagens de programação por usuários finais existentes, este trabalho apresenta uma nova proposta de programação por usuários finais para a especificação de agentes virtuais inteligentes. Demonstrando o comportamento através de um avatar imerso no ambiente virtual, o usuário poupa esforços na criação do conjunto de regras que determinam o comportamento do agente. Desta forma, espera-se facilitar o acesso de usuários finais ao desenvolvimento de ambientes virtuais mais realistas e difundir a utilização deste recurso.

Palavras-chave: Agentes Virtuais Inteligentes, Programação por Usuários Finais, Programação por Demonstração

Contatos: {genbarcam, vladimir.iorio}@gmail.com

1 Introdução

A diversidade de possibilidades oferecidas por ambientes virtuais (VEs, do inglês *Virtual Environments*) – simuladores, jogos, entre outros –, sem os riscos e custos inerentes de atividades no mundo real [Angros et al. 2002; Anastassakis et al. 2001], cria uma ótima opção para testar hipóteses e visualizar ideias. Melhor do que simplesmente olhar para símbolos, fórmulas e conceitos, trabalhar com esses ambientes permite ao usuário experimentar a realidade por trás destas definições.

Especialistas em suas respectivas áreas, com necessidades computacionais e que querem fazer sério uso dos computadores, mas que não estão interessados em tornarem-se programadores profissionais, referenciados na literatura como *Usuários Finais* [Nardi 1993], foram atraídos pelos benefícios da utilização de VEs. Surgiram então ambientes virtuais genéricos, sistemas adaptáveis e que, consequentemente, atenderiam a várias demandas.

A *Programação por Usuários Finais* (EUP, do inglês *End-User Programming*) [Myers et al. 2006; Cypher et al. 1993; Nardi 1993; Lieberman 2001], ao contrário da abordagem tradicional, que tenta fazer com que o usuário aprenda a representação aceita pelo computador, preza por tornar o sistema mais próximo do usuário, aceitando representações que não demandem tanta abstração. Buscando romper a barreira entre usuários finais e a especificação de sistemas de simulação, várias técnicas de EUP já foram utilizadas.

Os apelos visuais dos ambientes virtuais, através de gráficos super realistas, tornam-se pouco interessantes na falta de algo dinâmico e comportamental [Aylett and Cavazza 2001]. Agentes Virtuais Inteligentes (IVAs, do inglês *Intelligent Virtual Agents*) são agentes autônomos com representação física, parte de algum ambiente, e têm sido muito utilizados para dar mais realismo a VEs [Reynolds 1999; Brom 2006; Aylett and Cavazza 2001]. IVAs podem ser tutores em softwares educativos, inimigos e/ou aliados em um jogo ou mesmo personagens em animações [Brom 2006].

As técnicas de EUP exploradas para o desenvolvimento de VEs também foram aplicadas para facilitar a implementação de IVAs. O estudo da utilização das abordagens existentes, realizado neste trabalho, possibilitou a identificação dos pontos fortes e fracos dos mecanismos. Todavia, a especificação do comportamento de IVAs apresenta uma particularidade ainda não explorada pelos sistemas existentes. Depois da construção do ambiente, definição do avatar e das formas de interação dos IVAs entre si e com o VE, e representação das emoções do IVA, basta que o desenvolvedor especifique o comportamento dos personagens. Desta forma, percebe-se que a especificação do comportamento de IVAs pode ser feita utilizando exemplos demonstrados pelo usuário. Em [Reynolds 1999] a especificação do comportamento foi segmentada em (1) seleção de ações, (2) pilotagem e (3) locomoção. A abordagem deste trabalho é a camada de seleção de ações. Entende-se especificação do comportamento como a determinação das ações a serem executadas pelo IVA.

O objetivo deste trabalho é facilitar o acesso de usuários finais ao desenvolvimento de ambientes virtuais através da proposição de uma nova abordagem em EUP para a especificação do comportamento de IVAs. Além de agregar valor na definição de VEs mais interativos, os conceitos aplicados na proposta deste trabalho podem também ser transferidos para o mundo real. A programação de máquinas industriais ou mesmo de robôs domésticos, são exemplos de atividades que podem ser beneficiadas pela nova abordagem.

2 Trabalhos Relacionados

Várias técnicas de EUP já foram empregadas na busca pela facilitação da especificação do comportamento de IVAs por usuários finais. A configuração de parâmetros é a abordagem utilizada na criação de *bots* (abreviação de *robots*) no Counter Strike[1][Cole et al. 2004]. Esta técnica possibilita ao usuário ativar ou desativar opções que afetam o comportamento da aplicação fazendo uso somente de elementos da interface.

Linguagens de Script são comumente simplificações de linguagens de programação (LPs) tradicionais e possuem sintaxe personalizada para tratar os objetos e ações do domínio. Utilizada em muitos jogos e simuladores como: Robocode[2]; Gamebots[3] e Unreal Script que rodam sobre a plataforma do Unreal Tournament[4] (UT) [Cole et al. 2004]; e ambiente Logo [Kelleher and Pausch 2005].

As Linguagens Visuais Baseadas em Regras são aquelas nas quais o usuário determina as condições, e para cada uma destas, as ações associadas. O jogo Warcraft III[5] utiliza esta técnica [El-Nasr and Smith 2006]. A incorporação de mecanismos de programação por demonstração (PBD, do inglês *Programming by Demonstration*), em que o usuário utiliza os elementos visuais da aplicação para a criação dos condicionais é uma variante desta técnica. Agentsheets [Repenning 2000] e KidSim [Cypher and Smith 1995] são ambientes que a implementam.

Bimbo e Vicario em [Bimbo and Vicario 1995] e McDaniel e Myers em [McDaniel and Myers 1998] implementam soluções que utilizam a técnica Programação por Demonstração. Nesta, através do uso de técnicas de inteligência artificial, é criado um programa generalizado a partir de demonstrações realizadas pelo usuário.

O estudo das técnicas de EUP utilizadas para a especificação do comportamento de IVAs revela particularidades que comprometem o desempenho, ou mesmo restringem o acesso dos usuários finais. O número de parâmetros que se pode esperar que o usuário esteja

[1] Counter Strike (Valve Corporation) - www.counter-strike.net
[2] Robocode (Open Source) - robocode.sourceforge.net
[3] Gamebots (Open Source) - sourceforge.net/projects/gamebots
[4] Unreal Tournament (Epic Games) - www.unrealtournament3.com
[5] Warcraft III (Blizzard) - http://www.blizzard.com/us/war3/

apto a entender ao adotar a configuração de parâmetrose é bastante pequeno e consequentemente compromete a implementação de comportamentos que atendam às suas necessidades [Barbosa 1999]. O emprego de linguagens de script, ainda que muito mais simples que as LPs convencionais, assim como a utilização dos ícones propostos pelas linguagens de fluxo de controle, que não dispensam o entendimento da semântica da estrutura, necessitam de um mínimo conhecimento de programação que muitas vezes falta ao usuário final. Por sua vez, as LPs visuais baseadas em regras exigem a criação de condicionais basicamente para cada interação do agente com o meio, tornando a técnica inadequada para a especificação de comportamentos mais elaborados ou quando o agente precisa interagir com um ambiente complexo. Finalmente, os sistemas que incorporam a PBD restringem a atuação do usuário e demandam nova demonstração caso ocorram inferências errôneas nas exemplificações anteriores. Desta forma, vê-se que as técnicas de EUP utilizadas para a especificação do comportamento de IVAs são passíveis de questionamento quanto à real facilitação do trabalho do usuário. Portanto, percebe-se que mais pode ser feito no intuito de minimizar os problemas encontrados.

3 Programação do Comportamento de Agentes por Demonstração

A PBD é, por motivos óbvios, a mais adequada ao problema. Especificar comportamentos por demonstração é o mesmo que ensinar pelo exemplo. Todavia, o encapsulamento da solução gerada a partir da demonstração vai em desencontro com a usabilidade das abordagens que seguem esse mecanismo. Uma vez que o usuário não programa explicitamente, o comportamento inferido pelo sistema a partir da demonstração pode não estar de acordo com aquilo que era esperado, sendo necessário, portanto, realizar modificações na solução proposta. Nos sistemas encontrados na literatura, essas modificações são possíveis somente via nova demonstração. Tal solução, além de não economizar o tempo do usuário, pode causar insatisfação, uma vez que não assegura a correção do problema, expondo-o à necessidade de novo retrabalho.

As técnicas vistas isoladamente possuem pontos que as qualificam como alternativas para a solução do problema abordado neste trabalho. Contudo, existem também características que comprometem sua utilização. Observa-se, também, que a aplicação de mecanismos diferentes para resolução do problema não é conflitante, e sim complementar. Recursos seguindo a PBD resolvem a manipulação de variáveis textuais nas LPs baseada em regras, que por sua vez, permitem a modificação do comportamento, sem necessidade de nova demonstração.

Vista a complementaridade dos diferentes mecanismos, propõe-se a nova abordagem de EUP para a especificação do comportamento de IVAs. A programação visual de regras, com toda sua expressividade e seu fácil entendimento e manipulação, é a alternativa utilizada neste trabalho para contornar o encapsulamento das soluções geradas pela PBD.

Essa aliança caracteriza o caminho reverso da variante da programação visual de regras que utiliza os elementos visuais da aplicação para a criação dos condicionais. Ao invés de permitir que o usuário crie as regras exemplificando as situações que serão vividas no ambiente, nesta nova solução o usuário demonstra, através de um avatar imerso no VE, como o IVA deve se comportar. As ações tomadas pelo usuário diante das situações casualmente enfrentadas, resultam em regras de comportamento que serão seguidas pelo IVA, e nesta proposta podem ser posteriormente editadas sem nova demonstração. A abordagem permite que, por exemplo, um *bot* de um jogo de tiro de primeira pessoa seja gerado a partir do simples registro do jogo de um usuário.

A incorporação das LPs baseadas em regras à PBD traz consigo os problemas existentes neste mecanismo. Uma vez que as regras serão geradas automaticamente, não é necessário preocupar-se com o quão maçante seria criá-las, como discutido em [Coura et al. 2006], nem tão pouco com a possibilidade de geração de regras conflitantes. Contudo, sistemas que utilizam este mecanismo normalmente se atêm à especificação de comportamentos simples, sem muitas variáveis a serem consideradas, posto que uma grande quantidade de regras dificultaria aos usuários entender o que foi implementado. A proposta deste trabalho busca atender a todo o tipo de comportamento, simples ou complexo, e sobretudo garantir que usuários finais possam utilizá-la. Portanto, uma nova adaptação se fez necessária para assegurar a compreensão do usuário.

O conjunto de variáveis a ser considerado para a especificação de comportamentos complexos é muito extenso. Seguindo a ideia da configuração de parâmetros, este trabalho incorpora ao mecanismo de PBD os atributos que serão levados em conta para a criação das regras. Para cada atributo, também deve ser definido um conjunto padrão de valores, que pode ser posteriormente editado conforme o interesse do usuário. A coleção de atributos definidos estará disponível ao usuário, e a cada um estará associado seu respectivo elemento visual (se existir) e descrição. Antes de iniciar a demonstração o usuário deve definir quais atributos ele deseja que sejam considerados para a criação do comportamento. A ideia deste recurso é permitir que o usuário dose, conforme seu grau de entendimento, a quantidade de variáveis a serem manipuladas, permitindo maior predição do comportamento que será executado pelo IVA a partir da análise das regras geradas.

A quantidade de regras é outro fator crítico da solução proposta. Por serem geradas a partir de arranjos de atributos, o número de expressões condicionais resultante da geração automática é imenso. Visto o problema, além das regras foram também adotadas árvores de decisão como forma de representação do conhecimento. Os nós de cada nível representam um atributo e os ramos destes fazem o papel dos valores associados. Acredita-se que a utilização desta técnica pode contribuir muito para o melhor entendimento do usuário, uma vez que ao invés de ser exposta uma lista de regras, ainda que ordenadas, o usuário pode navegar entre os nós da árvore até alcançar aquela que represente a situação que ele deseja analisar. A técnica ainda agrega valor à solução ao impedir que sejam geradas expressões com diferença na quantidade de atributos testados, dispensando a necessidade de adoção de políticas de prioridade de execução para comportamentos vinculados a expressões mutuamente verdadeiras. Outro benefício está na avaliação das expressões condicionais. Enquanto regras não organizadas utilizando esta técnica precisam ser testadas uma a uma, algoritmos podem otimizar o posicionamento dos atributos nos níveis da árvore, de forma a definir qual o comportamento deve ser executado testando a menor quantidade possível de expressões.

3.1 Formalização da abordagem

A seleção de ações é o meio através do qual um IVA resolve o problema de decidir o que fazer, qual atitude tomar. É a parte executiva da inteligência dos agentes [Bryson 2004]. Sua análise permite predizer o comportamento de determinado agente sem ser necessário vê-lo agir. Segundo Bryson a adoção de padrões – planos reativos básicos[6], planos reativos POSH[7], máquinas de estado finitas e determinismo ambiental[8]– para a modelagem da camada de seleção de ações simplifica a solução e consequentemente facilita o entendimento do usuário.

Planos reativos básicos e POSH demandam noções de conceitos como prioridade e paralelismo, que podem não ser conhecidos pelo usuário e colocam em risco a compreensão da camada de seleção de ações. Máquinas de estado finitas oferecem uma abordagem mais palpável ao usuário. Todavia, por focar nas ações, possuem um conjunto de regras diferentes para cada estado, o que poderia dificultar a predição do comportamento que será executado. Determinismo ambiental, o padrão adotado neste trabalho, orienta-se nos eventos, discretiza o conjunto de situações que o agente pode presenciar e associa a cada uma delas a ação que deve ser tomada. A simplicidade do conjunto de regras, produto do determinismo ambiental, motivaram a escolha deste trabalho.

Tão importante quanto a escolha da PBD como estratégia de aprendizado, é a forma com que o sistema representa o conhecimento adquirido. A modelagem dos eventos como regras é intrínseca

[6] Basic Reactive Plans

[7] Parallel-rooted, Ordered, Slip-stack Hierarchical

[8] Environmental Determinism

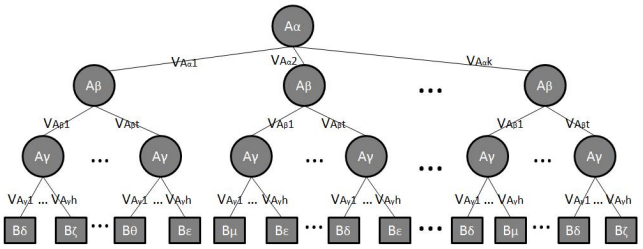


Figura 1: *Exemplo da camada de seleção de ações em esquema híbrido: regras e árvore de decisão.*

à adoção de LPs baseadas em regras, todavia, a quantidade de expressões condicionais motivaram a opção por um esquema híbrido: regras e árvores de decisão como representação do conhecimento presente na camada de seleção de ações.

Portanto, o comportamento gerado é formado pela tupla (A, B). Onde: A representa conjunto de atributos selecionados pelo usuário para a criação das regras, $A = \{A_1, A_2, \ldots, A_n\}$; e B, por sua vez, simboliza o conjunto de comportamentos demonstrados, $B = \{B_1, B_2, \ldots, B_k\}$. A é subconjunto de R, biblioteca de atributos que podem ser considerados pelo usuário e definidos previamente pelo desenvolvedor da aplicação. Cada atributo pertencente à biblioteca R possui seu conjunto de valores associado $V_{R_i} = \{V_{R_i1}, V_{R_i2}, \ldots, V_{R_im_{R_i}}\}$, que podem ser definidos pelo desenvolvedor ou editados conforme a necessidade do usuário. Não necessariamente, todos os comportamentos do conjunto B estarão inicialmente associados às conjunções entre os atributos, alguns deles podem estar disponíveis apenas para futura edição.

Um exemplo genérico do comportamento gerado pode ser visto na Figura 1. Neste caso, tem-se o conjunto de atributos selecionados pelo usuário sendo $A = \{A_\alpha, A_\beta, A_\gamma\}$ com seus respectivos conjuntos de valores: $V_{A_\alpha} = \{V_{A_\alpha 1}, V_{A_\alpha 2}, \ldots, V_{A_\alpha k}\}$, $V_{A_\beta} = \{V_{A_\beta 1}, \ldots, V_{A_\beta t}\}$ e $V_{A_\gamma} = \{V_{A_\gamma 1}, \ldots, V_{A_\gamma h}\}$. Os comportamentos exibidos na árvore: B_δ, B_ζ, B_θ, B_ϵ e B_μ pertencem ao conjunto B, mas não se pode dizer que B possui somente estes, outros podem estar disponíveis para posterior edição. A leitura de uma das expressões da árvore exemplificada é: Caso A_α seja igual a $V_{A_\alpha 1}$ e A_β igual a $V_{A_\beta 1}$ e A_γ igual a V_{A_γ}, então execute o comportamento B_δ.

4 Ambiente de testes

Embora existam diversos sistemas de criação de simulações genéricas, Agentsheets [Repenning 2000] e KidSim [Cypher and Smith 1995] por exemplo, a utilização destes para o desenvolvimento da plataforma de testes é impossível, pois seriam necessárias modificações e estes são proprietários. O custo de desenvolvimento de um VE que incorporasse os recursos visuais dos ambientes atuais é enorme. Entretanto, a modularização do desenvolvimento de jogos de computador [El-Nasr and Smith 2006] permite que estes sejam adaptados para atender às necessidades do trabalho.

Contudo, as adaptações que podem ser realizadas dependem das possibilidades oferecidas pelo jogo. El-Nasr e Smith, em [El-Nasr and Smith 2006], estudaram a utilização de várias linguagens de extensão e a *Unreal Script*, linguagem oferecida pelo UT, destacou-se por permitir a customização de praticamente todo o jogo. A expressividade da Unreal Script foi determinante para a escolha da plataforma de implementação da ferramenta de validação.

A extensa capacidade de personalização permitiu que o UT fosse utilizado para pesquisas em diversas áreas, e para diversos públicos. Na área de especificação de agentes virtuais inteligentes pode-se citar: Gamebots[Kaminka et al. 2002], Indigente[Monteiro 2006] e Pogamut 2[Burkert et al. 2007]. O primeiro permite o controle de personagens do UT em arquitetura cliente-servidor, enviando informações sensoriais para o cliente (que pode ser um IVA ou um jogador), e repassando ao servidor do jogo quais os comandos foram executados pelos agentes. Embora ofereça recursos para o desenvolvimento de agentes, a comunicação por mensagens, a quantidade de trabalho de baixo nível que precisa ser realizado antes de realmente iniciar o especificação dos IVAs, e a não existência de uma arquitetura de agentes pré-definida são barreiras mesmo para programadores mais experientes. Os demais, oferecem exatamente a solução para os problemas enfrentados no Gamebots. Ambos encapsulam-no, oferecendo uma camada de alto nível para a especificação dos IVAs, e uma arquitetura para a modelagem de agentes. A grande quantidade de informação disponível online, graças à vasta rede de usuários, o suporte por parte dos desenvolvedores e a continuidade do desenvolvimento, justificam a escolha do Pogamut 2 para a implementação da plataforma de testes.

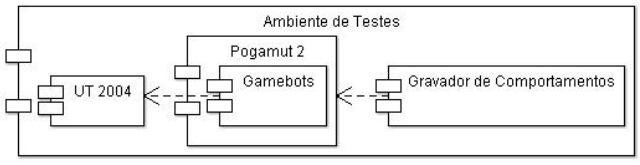


Figura 2: *Diagrama de componentes do ambiente de testes*

Portanto, a plataforma para a implementação do ambiente de testes da nova abordagem de EUP proposta será o jogo UT 2004, devido a sua flexibilidade de adaptação e o arcabouço Pogamut 2, justificado acima. O diagrama de componentes do ambiente de testes é apresentado na Figura 2.

4.1 Vista Geral do Funcionamento do Sistema

A ideia básica da aplicação a ser implementada é permitir que o usuário jogue UT 2004 e que a partir do comportamento por ele executado, um IVA seja gerado. Na interface do ambiente de testes, o usuário se conectará ao servidor de um jogo previamente criado no UT. Neste, será inserido um agente marionete, que será comandado pelo usuário diante das situações vividas no ambiente. Deverá ser solicitada, então, a geração das regras a partir do registro da demonstração. A camada de seleção de ações será disponibilizada e o usuário terá a opção de editá-la. Satisfeito com as expressões resultantes, restará conectar o IVA padrão no ambiente, que interpretará as regras geradas e executará um comportamento semelhante ao que foi demonstrado.

5 Comparativo com outras técnicas

Buscando apresentar os benefícios da proposta deste trabalho em relação a outras técnicas de programação para usuários finais já utilizadas para o mesmo fim, discute-se hipoteticamente a criação do comportamento de IVAs através da configuração de parâmetros, LPs visuais baseadas em regras e programação por demonstração. Embora as linguagens de script tenham, também, sido apresentadas como opções já disponibilizadas aos usuários finais para a realização da tarefa, estas estão muito aquém da facilitação das demais e, por isso, foram preteridas nesta análise.

Em sistemas que utilizam a configuração de parâmetros quanto maior o número de regras mais realista tende a ser o agente e mais tempo será demandado para a implementação. Fica claro que esta alternativa não passa da disponibilização da camada de seleção de ações ao usuário final, também oferecida neste trabalho. Todavia, as customizações se resumem aos valores comparados nas expressões condicionais. Não é possível alterar a quantidade de regras, o que pode dificultar o entendimento, como já discutido anteriormente. Outro problema é a impossibilidade de criar novas regras com ações distintas, ou mesmo editar as regras pré-estabelecidas. Portanto, a configuração de parâmetros é nada além da possibilidade de alterar o conjunto padrão de valores de cada um dos atributos disponibilizado na biblioteca do estudo de caso, recurso que, embora não implementado, já faz parte da proposta deste trabalho. Logo, percebe-se que, a não necessidade de interferência manual do usuário em todas as regras, evitada pela demonstração, e a possibilidade de alteração das ações associadas às regras são melhorias relevantes da nova abordagem em relação a configuração de parâmetros.

As LPs visuais baseadas em regras, principalmente as que utilizam os elementos visuais da aplicação para a criação dos condi-

cionais, são as mais utilizadas na literatura para a especificação do comportamento de agentes. Por basearem-se na interface do VE no qual interagem, são mais intuitivas e consequentemente mais acessíveis aos usuários finais. Tradicionalmente, as LPs visuais baseadas em regras trabalham com a interação do agente com o meio sem a necessidade de analisar propriedades internas do mesmo. No caso da criação de um IVA, tal qual o que será desenvolvido neste estudo de caso, a análise de atributos é indispensável. Então, a criação das regras envolveria a análise dos que serão disponibilizados na biblioteca, cada qual com três faixas de valores em média. A quantidade de regras a ser gerada é imensa, o que torna a técnica inadequada, uma vez que seria maçante ao usuário ter de criar todo o conjunto de expressões manualmente. Não obstante, a regras estariam organizadas sequencialmente, dificultando o entendimento e também a eficiência da execução. Portanto, graças a possibilidade de demonstrar o comportamento, a técnica desenvolvida neste trabalho é muito mais eficiente, e devido a organização da camada de seleção de ações em árvore, mais compreensível.

Por sua vez, a criação de comportamentos através da programação por demonstração é a que mais se aproxima à técnica desenvolvida neste trabalho. Em relação às propostas implementadas em [Bimbo and Vicario 1995] e [McDaniel and Myers 1998], o não ocultamento da camada de seleção de ações, permitindo criação, ou edição, de regras sem necessidade de nova demonstração, e a possibilidade de demonstrar o comportamento jogando, e não em situações específicas ou ciclos, incorporando o fator diversão à tarefa, agregam valor à abordagem deste trabalho.

6 Conclusões e Trabalhos Futuros

Embora a especificação do comportamento do IVAs através da demonstração já tenha sido proposta na literatura, o estudo realizado neste trabalho das diversas abordagens de EUP, voltadas, ou não, à tarefa em questão, possibilitou adaptações à técnica existente. Estas fazem com que a nova proposta, comparada, hipoteticamente, aos mecanismos já utilizados para tal propósito, facilite ainda mais o acesso a sistemas de simulação a profissionais de diversas áreas, antes privadas dos benefícios desta tecnologia.

Ainda que o trabalho consista na proposta de uma nova abordagem de programação para usuários finais, a implementação do ambiente de testes é fundamental para a realização de testes e consequentes análises comparativas quantitativas em relação à satisfação e desempenho do usuário na utilização desta e das outras técnicas já aplicadas à mesma tarefa. Pretende-se desenvolver a aplicação como parte da interface de desenvolvimento integrado do Pogamut 2. Desta forma sua utilização seria difundida e desenvolvimento partilhado junto a àqueles que faz uso do arcabouço, possibilitando maior contribuição do trabalho à comunidade científica.

Agradecimentos

À CAPES e FAPEMIG, pelo apoio financeiro.

Referências

ANASTASSAKIS, G., RITCHINGS, T., AND PANAYIOTOPOULOS, T. 2001. Multi-agent systems as intelligent virtual environments. In *KI '01: Proceedings of the Joint German/Austrian Conference on AI*, Springer-Verlag, UK, 381–395.

ANGROS, JR., R., JOHNSON, W. L., RICKEL, J., AND SCHOLER, A. 2002. Learning domain knowledge for teaching procedural skills. In *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, ACM, New York, NY, USA, 1372–1378.

AYLETT, R., AND CAVAZZA, M. 2001. Intelligent virtual environments - a state-of-the-art report. In *Eurographics'01*.

BARBOSA, S. D. J. 1999. *Programação via Interface*. PhD thesis, Departamento de Informática–Pontifícia Universidade Católica do Rio de Janeiro.

BIMBO, A. D., AND VICARIO, E. 1995. Specification by-example of virtual agents behavior. *IEEE Transactions on Visualization and Computer Graphics 1*, 4, 350–360.

BROM, C., 2006. Action selection for virtual humans in large environments. Doctoral Thesis Abstract – Faculty of Mathematics and Physics – Department of Software and Computer Science Education – Charles University in Prague.

BRYSON, J. J. 2004. Action selection and individuation in agent based modelling. In *Proceedings of Agent 2003: Challenges in Social Simulation*, Argonne National Laboratory, 317–330.

BURKERT, O., KADLEC, R., GEMROT, J., BÍDA, M., HAVLÍČEK, J., DÖRFLER, M., AND BROM, C. 2007. Towards fast prototyping of ivas behavior: Pogamut 2. In *IVA '07: Proceedings of the 7th international conference on Intelligent Virtual Agents*, Springer-Verlag, Berlin, Heidelberg, 362–363.

COLE, N., LOUIS, S. J., AND MILES, C. 2004. Using a genetic algorithm to tune first-person shooter bots. In *Proceedings of the IEEE Congress on Evolutionary Computation*, IEEE, New York, NY, USA, vol. 1, 139–145.

COURA, D. P., IORIO, V. O. D., LIMA, A. G., OLIVEIRA, A. P., AND ANDRADE, M. V. A. 2006. Animaçoes através de programação por demonstração. In *IHC '06: Proceedings of VII Brazilian symposium on Human factors in computing systems*, ACM, New York, NY, USA, 81–90.

CYPHER, A., AND SMITH, D. C. 1995. Kidsim: end user programming of simulations. In *CHI '95: Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM Press/Addison-Wesley Publishing Co., NY, USA, 27–34.

CYPHER, A., HALBERT, D. C., KURLANDER, D., LIEBERMAN, H., MAULSBY, D., MYERS, B. A., AND TURRANSKY, A., Eds. 1993. *Watch what I do: programming by demonstration*. MIT Press, Cambridge, MA, USA.

EL-NASR, M. S., AND SMITH, B. K. 2006. Learning through game modding. *Comput. Entertain. 4*, 1.

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.

KELLEHER, C., AND PAUSCH, R. 2005. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Comput. Surv. 37*, 2.

LIEBERMAN, H., Ed. 2001. *Your wish is my command: programming by example*. Morgan Kaufmann Publishers Inc., CA, USA.

MCDANIEL, R. G., AND MYERS, B. A. 1998. Building applications using only demonstration. In *IUI '98: Proceedings of the 3rd international conference on Intelligent user interfaces*, ACM, New York, NY, USA, 109–116.

MONTEIRO, I. M., 2006. IAF - indigente agent framework: Um framework para o desenvolvimento de agentes cognitivos em jogos de primeira pessoa. Monografia de Graduação – Depto. de Ciência da Computação – Instituto de Matemática – UFBA.

MYERS, B. A., KO, A. J., AND BURNETT, M. M. 2006. Invited research overview: end-user programming. In *CHI '06: CHI '06 extended abstracts on Human factors in computing systems*, ACM, New York, NY, USA, 75–80.

NARDI, B. A. 1993. *A small matter of programming: perspectives on end user computing*. MIT Press, Cambridge, MA, USA.

REPENNING, A. 2000. Agentsheets: an interactive simulation environment with end-user programmable agents. In *Proceedings of the IFIP Conference on Human Computer Interaction*, 1–8.

REYNOLDS, C. 1999. Steering behaviors for autonomous characters. In *Proceedings of the Game Developers Conference 1999*, Miller Freeman Game Group, 763–782.

Evolução de Estratégias em Unidades Controladas Pelo Computador em Jogos de Estratégia em Tempo Real Com a Utilização de Algoritmos Genéticos

Thiago D. Mendonça Christiano L. Santos

Universidade Federal de Sergipe, Departamento de Computação, Brasil

Abstract

One of the problems faced by real-time strategy (RTS) games is the predictability of the actions made by computer-controlled opponents, due to the use of artificial intelligence techniques that do not permit the machine to learn dynamically. This article aims to propose the use of genetic algorithms in RTS games to model an evolutive strategy, as well present GenCraft, an application developed as a case study.

Keywords: Artificial Intelligence, Real-Time Strategy, Genetic Algorithms

Resumo

Um dos problemas enfrentados por jogos de estratégia atualmente é a respeito da previsibilidade das ações dos oponentes controlados pelo computador, devido ao emprego de técnicas de inteligência artificial que não permitem o aprendizado de forma dinâmica. Este artigo tem como objetivo propor a modelagem de uma estratégia evolutiva em unidades controladas pelo computador em jogos de estratégia em tempo real com a utilização de algoritmos genéticos, bem como apresentar o GenCraft, uma aplicação desenvolvida como estudo de caso.

Palavras Chave: Inteligência Artificial, Estratégia em Tempo Real, Algoritmos Genéticos

Contato dos Autores:
thiagodm@dcomp.ufs.br
christianolimasantos@yahoo.com.br

1. Introdução

Os jogos digitais atuais têm se tornado cada vez mais realistas, especialmente em termos de apresentação gráfica do mundo virtual no qual o jogo é situado. Para aumentar o realismo nos jogos, o próximo passo é fazer com que os personagens desses mundos virtuais tornem-se aptos a responder efetivamente às ações do jogador [Ponsen et al. 2006].

O campo de inteligência artificial (IA) em jogos existe desde o aparecimento dos jogos digitais nos anos 70. A percepção pública da IA em jogos, mesmo nos mais atuais, remete a jogos antigos como *Pac-Man*, onde predominam ações repetitivas [Rabin 2002].

Com o advento dos computadores pessoais cada vez mais rápidos, jogos interativos ficaram cada vez mais populares. Alguns exemplos desses jogos são os de estratégia em tempo real onde os jogadores comandam exércitos que se confrontam. Para esses jogos há uma grande demanda de um comportamento inteligente que seja capaz de tomar decisões satisfatórias e em tempo real. Entretanto, mesmo em jogos RTS (*Real-Time Strategy*), como *Starcraft* e *Warcraft* da *Blizzard Entertainment* e *Age of Empires* da *Ensemble Studios*, a performance da Inteligência Artificial é pobre quando comparada aos padrões humanos, onde o computador executa estratégias repetitivas e fáceis de combater [Buro e Furtak 2004].

Desta forma, há uma preocupação crescente com a criação de agentes inteligentes com melhor capacidade de se adaptar e evoluir durante uma partida em jogos eletrônicos. Isso nos leva a um dos grandes desafios atuais das empresas desenvolvedoras de jogos eletrônicos, que é proporcionar ao jogador uma ambiente não somente competitivo, mas adaptável às escolhas do jogador, como citam os autores em [Schadd et al. 2007], que um fator importante para a escolha de uma estratégia é a própria estratégia do oponente, conseguindo assim uma situação desafiadora capaz de oferecer uma satisfação maior com o jogo.

Para isso, propõe-se um abordagem com algoritmos genéticos, permitindo a evolução da IA envolvida. Devido à sua natureza evolutiva, os algoritmos genéticos podem ser muito bem empregados em aplicações que exijam muitas interações entre diversos agentes em um mesmo ambiente, como é o caso dos jogos de estratégia, onde há muitas unidades e civilizações, evoluindo com o tempo e a importância da adaptação de estratégia e tomada de novas decisões é fundamental para garantir a experiência do usuário.

O maior desafio no uso de algoritmos genéticos em jogos RTS é o tempo necessário para o aprendizado. O algoritmo pode demorar muito tempo para convergir para uma boa estratégia, fazendo com que o jogador não sinta uma grande dificuldade no decorrer do jogo.

Este artigo propõe a investigação e utilização de algoritmos genéticos em jogos RTS para modelar uma estratégia evolutiva em unidades controladas pelo computador, utilizando como estudo de caso o jogo GenCraft, desenvolvido durante a fase de pesquisa, discutindo os resultados obtidos.

2. Trabalhos Relacionados

[Ponsen e Spronck 2004] propõem um modelo de aplicação de algoritmos evolutivos em jogos RTS bastante similar ao proposto neste trabalho. Eles perceberam que as rotinas comuns de criação de estratégias para unidades combatentes eram deficientes ao enfrentar oponentes com estratégias mais otimizadas. Um jogo de estratégia equipado com uma IA evolutiva pode potencialmente ignorar estratégias comuns e ser criativo.

Os autores utilizaram *Wargus*, um jogo RTS similar a *Warcraft II*, como estudo de caso e programaram rotinas de algoritmos evolutivos para as unidades do jogo. Obtiveram questionamentos muito semelhantes aos discutidos neste trabalho, como escolha do tamanho ideal da população. Para treinar essa população, os autores utilizaram um oponente controlado pelo computador.

Este artigo visa contribuir com o estado da arte ao criar um ambiente similar ao descrito, porém com a diferença que a aplicação seja "jogável", investigando assim a utilização das técnicas em um jogo real interativo.

3. Algoritmos Genéticos

No desenvolvimento de jogos eletrônicos, técnicas de inteligência artificial normalmente são utilizadas. Dentre elas têm-se algoritmos de *pathfinding*, sistemas baseados em regras, agentes inteligentes, entre outros.

Uma técnica atualmente subutilizada em jogos digitais é o uso de algoritmos genéticos na construção do comportamento inteligente do oponente.

Os algoritmos genéticos possuem uma estrutura básica comum [Lecheta 2004]:

1. Geração aleatória de indivíduos (somente na primeira geração);
2. Seleção de indivíduos para realizar reprodução (*crossover*) aplicando-se a função *fitness* para calcular seu nível de adaptação ao meio;
3. Imposição de variações aleatórias nos cromossomos dos indivíduos resultantes (mutações);
4. Reinício do processo com os novos indivíduos.

Ao final da aplicação do algoritmo, obtêm-se um ou mais indivíduos evoluídos, ou seja, mais aptos ao ambiente.

[Dalmau 2004] cita que os algoritmos genéticos em jogos podem ser usados para a geração de uma população, criando indivíduos com diferentes "cargas genéticas", que podem influenciar seu comportamento. Essa técnica pode ser utilizada para simular gerações de unidades combatentes num ambiente de um jogo de estratégia. Dessa forma, o comportamento inteligente estaria sendo aplicado diretamente nas unidades, ou seja, cada combatente teria seu próprio comportamento autônomo e "inteligente".

Esse efeito pôde ser observado em GenCraft, onde cada unidade possui autonomia sobre suas ações, que por sua vez, são definidas de acordo com os genes de cada unidade.

4. Estudo de Caso - GenCraft

Trata-se de um jogo, ilustrado na Figura 1, onde o usuário comanda uma civilização e pode tanto recrutar unidades construtoras e combatentes, como construir depósitos para coletar recursos ou quartéis para recrutar mais unidades combatentes.

Figura 1: GenCraft

O jogo possui as seguintes unidades:

- Peão – unidade básica, não possui poder de ataque e possui poucos pontos de vida (PVs), porém constrói depósitos e quartéis;
- Orc – unidade combatente, possui muitos PVs e alto poder de ataque, porém pode atacar outras unidades apenas a curta distância;
- Elfo – unidade combatente, possui moderados PVs e baixo poder de ataque, porém pode atacar à distância.

Além disso, o jogador possui à sua disposição as seguintes construções:

- Base – construção que possibilita recrutar peões. Se destruída, o jogador proprietário da mesma perde o jogo, tornando-se este o objetivo central do mesmo;
- Depósito – construção que, se construída próxima a recursos, começa a coletá-los;
- Quartel – construção que recruta unidades combatentes.

Conforme os principais jogos estratégicos de guerra, recursos são necessários para o recrutamento de novas unidades bem como construção de novas

estruturas. Em GenCraft, dois são os recursos necessários para tal desenvolvimento:

- Madeira – recurso necessário para a criação de construções. Coletado de florestas no mapa;
- Ouro – recurso necessário para o recrutamento de unidades. Coletado de minas no mapa.

Em GenCraft, algoritmos genéticos são empregados na criação de novas unidades combatentes controladas pelo computador a fim de determinar seus comportamentos. Desta forma, a partir de indivíduos inicialmente criados aleatoriamente, espera-se que os mesmos alterem seus comportamentos a fim de melhor agirem no ambiente do jogo.

4.1 Arquitetura do Cromossomo

Em GenCraft, somente as unidades combatentes, ou seja, os orcs e os elfos, são criados a partir de técnicas de algoritmo genético.

Para que possa, de fato, haver um melhoramento no comportamento das unidades no decorrer de uma partida, os genes que compõem o cromossomo de cada indivíduo devem conter atributos que possam ser utilizados para calcular a estratégia final da unidade. No estudo de caso, utilizamos sete atributos, denominados a_0, a_1, a_2, a_3, b_0, b_1 e b_2.

Os atributos de a_0 a a_3 são parâmetros para o cálculo do nível de ameaça que cada unidade inimiga oferece à unidade portadora deste gene. Já os atributos de b_0 a b_2 determinam qual estratégia será escolhida pela unidade.

Existem duas estratégias básicas implementadas em cada unidade:

- Atacar – A unidade que decide pela estratégia de atacar determina qual unidade inimiga causa maior ameaça e ataca-a;
- Recuar – A unidade que decide pela estratégia de recuar movimenta-se em direção às proximidades de sua base para refugiar-se.

Os atributos possuem valores inteiros. Segue uma descrição mais detalhada dos genes das unidades, assim como sua faixa de valores:

- a_0 – influência que os PVs de uma unidade inimiga sobre a ameaça que esta causa à esta unidade (0-100);
- a_1 – influência que a quantidade de unidades inimigas ao redor de uma certa unidade inimiga tem na ameaça que esta causa à esta unidade (0-100);
- a_2 – influência do fato de uma certa unidade inimiga ser uma criatura (elfo, orc ou peão) ou uma construção (base, depósito ou quartel) na ameaça que esta causa à esta unidade (0-100);
- a_3 – influência da proximidade de uma certa unidade inimiga tem na ameaça que esta causa à esta unidade (0-100);
- b_0 – quantidade mínima de PVs que a unidade deve ter para decidir pela estratégia de atacar (0-70);
- b_1 – quantidade mínima de unidades aliadas em sua proximidade (reforços) que a unidade deve ter para decidir pela estratégia de atacar (0-5);
- b_2 – quantidade mínima de unidades aliadas a mais que unidades inimigas em todo o campo que a unidade deve ter para decidir pela estratégia de atacar (0-3).

Em cima desses atributos será feito um cálculo que irá determinar finalmente a estratégia tomada pela unidade neste momento até que seu objetivo (destruir o inimigo alvo ou recuar com sucesso à base) seja cumprido ou o ambiente mude drasticamente (seus pontos de vida diminuam, o inimigo recrute muitas unidades, seus aliados morram, etc.), quando, então, a unidade deverá perceber o ambiente novamente para decidir sua nova estratégia.

Quando, uma unidade morre, calcula-se para ela um valor de *fitness*, que indica a probabilidade dessa unidade ser escolhida para passar seus genes para a próxima geração. Para o cálculo do *fitness*, levamos em conta vários fatores – tempo de vida da unidade, quantidade de ataques que ela desferiu e de construções e criaturas que ela derrotou.

Para a implementação do algoritmo genético, foi utilizada seleção por roleta, *crossover* por cruzamento uniforme, mutação por inversão de bit e não foi implementado elitismo.

4.2 Resultados

A partir de discussões sobre o tempo de jogo necessário para o aprendizado do oponente e a quantidade de gerações necessária para tal, resolveu-se utilizar o valor de cinco indivíduos por geração. Isso significa que a cada cinco unidades que morrem e têm seu *fitness* calculado é iniciada uma nova iteração, e essas mesmas cinco unidades farão parte da seleção para propagar seus genes, e assim por diante.

Foram realizadas algumas partidas entre um jogador humano e o computador, utilizando os parâmetros já definidos para os algoritmos genéticos, gravando-se os logs das partidas para interpretação dos resultados. Escolheu-se paralisar os jogos com trinta e cinco minutos de partida, como um valor limite para a observação do aprendizado do oponente.

Durante os primeiros minutos de jogo, em geral percebia-se que as unidades adotavam estratégias aleatórias e pouco inteligentes, recuando ou atacando quando desnecessário. Após, em média, quinze minutos, havia alguma mudança no comportamento

geral. Algumas unidades ainda apresentavam comportamentos não muito eficientes, entretanto defendiam prontamente sua base quando unidades inimigas aproximavam-se.

Após os trinta e cinco minutos de jogo, paralisava-se para analisar os resultados. Em termos de jogabilidade, nessa etapa da partida, as unidades já apresentavam um melhor comportamento, atacando em momentos propícios, apesar de haver ainda certa quantidade de unidades optando somente pela tática de recuar. A Figura 2 demonstra um gráfico com a evolução da média dos valores dos genes que obtiveram uma mudança mais significativa durante os testes.

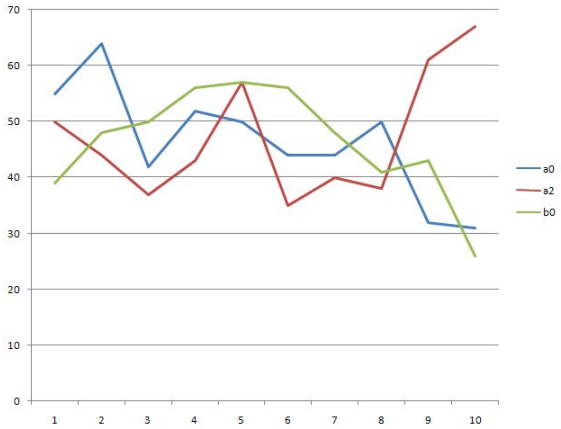


Figura 2: Gráfico de Atributos

No eixo horizontal estão as gerações (da 1ª à 10ª) e no eixo vertical estão os valores absolutos dos atributos em questão. A linha azul corresponde ao atributo a_0, a vermelha ao atributo a_2 e a verde a b_0.

Percebe-se que as unidades optaram por dar pouca importância aos PVs do inimigo ao atacar (a_0), como também decidiram que construções causam muito mais ameaça do que criaturas (a_2) e, por fim, deram pouca importância à quantidade de PVs necessária para escolher por atacar (b_0). Isso demonstra que as unidades decidiam atacar com muito mais frequência.

Demonstra-se assim a adaptação das unidades às circunstâncias do jogo, a ocorrência do aprendizado e a eficácia do método em tempo hábil, requisitos essenciais para algoritmos de inteligência artificial aplicados a jogos.

5. Considerações Finais

Este trabalho apresentou-se como o primeiro passo para a investigação de soluções mais robustas no campo de desenvolvimento de jogos eletrônicos e, mais especificamente, jogos de estratégia em tempo real. Por meio deste, confirmou-se a aplicabilidade dos algoritmos genéticos.

Por explorar uma área ainda pouco visada no mercado atual de jogos, é válida a tentativa de fomentar a preocupação com a criação de uma IA mais flexível e com um maior poder de evolução durante o jogo.

6. Futuras Extensões

Como futura extensão, em GenCraft, pretende-se investigar outras técnicas de IA como *dynamic scripting* para diversificar e melhorar o comportamento do oponente controlado pelo computador, além de criar estratégias mais complexas com a finalidade de conseguir resultados mais contundentes, realizando testes também com jogadores humanos com estratégias variadas, com o intuito também de verificar a validade de suportar oponentes aleatórios no início da partida.

Pretende-se ainda aperfeiçoar o experimento, expandindo-o para SPRING, uma *engine open source* para jogos RTS, com o objetivo de melhorar a jogabilidade, aumentar a complexidade das unidades e principalmente para poder realizar testes mais eficientes e comparar os resultados com outras IAs desenvolvidas por outros usuários.

Referências

BURO, M. E FURTAK, T., 2004. RTS Games and Real-Time AI Research. *Em: Advances in Computer Games X, pp. 159-174. Kluwer Academic Press*.

DALMAU, D., 2004. Core Techniques and Algorithms in Game Programming. *Indianapolis, New Riders.*

LECHETA, E., 2004. Algoritmos Genéticos para Planejamento em Inteligência Artificial. *Tese de Mestrado, Setor de Ciências Exatas, Universidade Federal do Paraná, Curitiba, Brasil.*

PONSEN, M., MUÑOZ,-AVILA, H., SPRONCK, P. E AHA D., 2006. Automatically Generating Game Tactics through Evolutionary Learning, *Em: AI Magazine Vol. 27 Nº 3, pp. 75-84.*

PONSEN, M., SPRONCK P., 2004. Improving Adaptive Game AI with Evolutionary Learning, *Em: Computer Games: Artificial Intelligence, Design and Education (CGAIDE 2004). pp. 389-396, University of Wolverhampton.*

RABIN, S., 2002. AI Programming Wisdom. *Hingham, MA, USA: Charles River Media, Inc.*

SCHADD, F., BAKKER, S. E SPRONCK, P., 2007. Opponent Modeling in Real-Time Strategy Games. *Em: 8th International Conference on Intelligent Games and Simulation (GAME-ON 2007), M. Rocetti, Ed.2007, pp. 61-68.*

SPRING. [online] Disponìvel em: http://springrts.com/ [Acessado em 4 de setembro de 2009].

Game Architecture for Business Simulation Games in XNA: The VTeam case

Bruno Jamir e Silva Raphael L. B. de Barros Diogo C. Lemos Danielle R. D. da Silva
Patricia R. Tedesco Geber L. Ramalho

Universidade Federal de Pernambuco

Abstract

Business simulation is a kind of video game aimed at reproducing a real business environment where the player plays the role of a manager, making decisions in order to make the business prosper. However, in several related researches about game development and architecture definition, business simulation games are rarely mentioned. Thus, this paper presents a core architecture for business simulation games, based on their common requirements and interface features. The proposed architecture was applied in a real game development project in order to validate it.

Keywords: Business, Simulation, Game, XNA, Architecture

Authors' contact:
brunojamir@gmail.com,{rlbb, dcl, drds, pcart, glr}@cin.ufpe.br

1. Introduction

Video games are no longer an exclusive hobby for children and teenagers. Every year, games interest more adults [Esa 2009]. One particular kind of game that attracts this new public is *business simulation games*. These are designed to simulate a real business environment where the player plays the role of the manager, making decisions in order to make the business prosper [Laramée 2002]. To attract this public, these games are designed to keep the familiarity with real world tools, such as management software and information systems.

Although there are several related works about game development process and architecture definition, business simulation games are rarely mentioned. This paper presents a core architecture for business simulation games, based on common requirements and interface features present in this kind of game. The proposed architecture main goal is to simplify the development process of business simulation games, providing scalable and efficient core features.

To validate the proposed architecture, a business simulation game called VTeam was developed over it [VTeam 2009]. This game simulates a software development environment, and the player plays the role of project manager. The player can attribute tasks, resolve conflicts between the team members, give some feedback to development team, etc. To provide a believable simulation, the game characters were implemented as synthetic actors [Silva 2009].

This paper is organized as follows. Section 2 explains the main business simulation games features. Section 3 shows the proposed architecture for business simulation games. Section 4 details the application of the proposed architecture in the development of a real game. Finally, section 5 our conclusions and suggestions for further work.

2. Business Simulation Games

Business simulation games focus on economic process management using a simplification of the business environment. These games can be described as construction simulations, when they focus on building elements to achieve a goal, or management simulation, when the game elements are already set and the actions are focused on resource management [Laramée 2002].

Business simulation games often share similar features, related to how the information is displayed and how the way the player interacts. Thus, for the development of business simulation games the following requirements must be met:

[R1] Mouse based actions: since all actions of the game are triggered by the mouse cursor, the player should be able to perform actions of selection, click and drag & drop intuitively with an immediate visual response.

[R2] Multiple (simultaneous) screens: the large number of actions available to the user and the need to display different groups of information requires an intelligent grouping of interface items. To address this requirement, the game must provide the player with several sub-screens that can be accessed from the main game screen.

[R3] Several interface components: the player controls the system through interface items arranged in the screens, such as buttons, checkboxes and tab panels. Thus a standardization of these elements is necessary in order to both simplify the development and maintain the usability cohesion of the system as a whole.

[R4] Base Simulation: in such games a large amount of calculation is processed in the background to simulate a real environment. Such calculations can be related to economics, statistics, and artificial intelligence formulas. This should be executed in parallel to the interface processes, in order to avoid jeopardizing the visual performance.

3. XNA Architecture for Business Simulation Games

Based on business simulation games' features and the architectural pattern offered by XNA Framework starter kits, a general architecture for business simulation games will be proposed in this section. Such architecture consists of 3 main modules: the graphics module, the services module, and the content loading module. The first component is the responsible for displaying the game elements to player. In other words, it controls the screen presentation, and the drawing of its visible elements. It is composed by screens, user interface components (like buttons, text boxes, etc.) and simulation views. These are specific visualizations of the game's simulated world.

The services module is responsible for providing resources for the game, helping in its implementation. The components of the services module were called managers. These managers are well-defined and almost independent modules. Thus, these managers respond to specific requests of the game's screens and their components. Examples of managers include: audio manager, input manager, and simulation manager, that simulate the game rules and artificial intelligence.

Finally, the content loading module is responsible for loading the resources that will be used in the game. It is responsible for loading image files, sound files and XML files for game configurations. The content loading module is composed by well-defined responsibility elements. Thus, an element may be the loader of character's animations, while another is responsible for loading the map that defines the character's positions in the world of game, for example.

In Figure 1, it is possible to correlate the created architecture with the requirements of the simulation games, mentioned in section 2 of this paper. Therefore, the way by which the requirements were covered by the architecture is explained below:

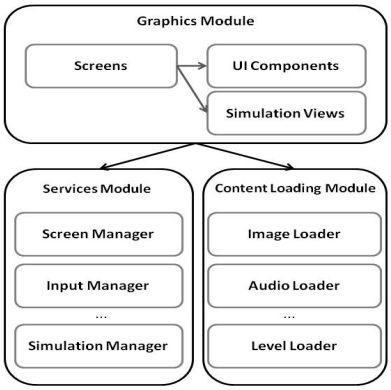


Figure 1: Business Simulation Game Architecture.

[R1] Mouse based actions: the interaction in a business simulation game is mostly done with the mouse. Therefore, the way the input is treated should be simple and scalable. Instead of pooling, we used an approach with events and delegates, available in the .NET Framework. In this approach, the components do not make consecutive requests for an input change. Instead they are registered as listeners of input events, raised by a sender class. For example, if the player selects the "New Game" option in the main screen, the mouse click action will be captured by the input manager and passed to the active screen. Upon receiving this event, the screen will decide what action to take.

[R2] Multiple (simultaneous) screens: to control frequent transitions between the game screens a Screen Manager was developed. This class is responsible for the resource allocation (and deallocation) of involved screens, and for controlling the effects of screen transitions.

[R3] Different interface components: business simulation games have an interface similar to general-purpose applications. Therefore, it is necessary to provide user interface components such as buttons, menus, and text boxes. To meet this requirement, the proposed architecture has these graphical components at the graphics module. However, there are some graphical objects that cannot be represented by simple interface components (e.g. a window of the world filled with characters). In this architecture these components will be considered as Simulation Views.

[R4] Base Simulation: to simulate the game rules, some managers can be developed in the service module. For example, if the game aims to simulate financial transactions, it is possible to create a manager that simulates the behavior of a stock exchange.

These requirements are found in most business simulation games. In order to validate the consistency of the proposed architecture, we have developed the

VTeam game, which simulates a software development environment. In this game, the player plays the role of team manager. The details concerning the architecture created for VTeam will be explained in the next section.

4. Business Simulation Game Architecture: The VTeam Case

A good example of use of Business Simulation games is in serious games. These games have been adopted in various universities as a support tool for learning business management. For instance, the games Virtual-U [Virtual University 2009] and Capitalism [Capitalism 1996] are games adopted at MBA courses to learn the complex process to manage and to build a business. This is also proposed by the Virtual Team Project that has been developing the VTeam business game.

5.1 The Game VTeam

Virtual Team (VTeam, in short) is a business simulation game developed with the purpose of training project managers, focusing mainly on people management and human resources. This game is an extension of the prototype developed previously by the Smartsim project [Smartsim 2006] including new features and business conflict scenarios to be learned by the player. VTeam's goal is to provide a greater experience of organizational, methodological, cultural and personal processes for the manager being trained.

This game offers to both instructor and player a richer experimentation scenario when compared to traditional methods and simulators used to teach those concepts. Thus, in order to provide a good simulation and give the player a proper learning experience, the VTeam game flow covers the following phases:

- *Planning*: a project plan is presented to the player. This plan is composed by a set of tasks, an initial budget, and some goals to be achieved. The player also has to contract the team selecting the characters to fill in the roles of programmer, architect and analyst.
- *Start up:* at this phase the player can interfere directly with the project through the environment were the characters interact with each other. The player can provide feedback, give rewards and assign project tasks for each of the characters from this phase on.
- *Development:* the player must follow the development of the tasks and solve conflicts that may occur between the characters. He has also to keep the client informed of the project status and interact with the team in order to optimize their job.
- *Finish:* once all the tasks are done and the goal is achieved (or the project time and budget are over), the game ends and a detailed result screen is presented, providing information allowing both the trainer and the trainee to reflect over the game performance.

Based on the presented requirements (in section 2) it was possible to apply the proposed architecture in VTeam game. Some details of this instantiation will be presented in next sections.

5.2 VTeam Game Screens

Each of the screens contains a set of user interface components grouped on a panel. Each screen is responsible for drawing and controlling the input of these components. Besides, the screens control the navigation between themselves requesting the *ScreenManager* to change the active current screen. Figure 2 shows some interface components at the character status game screen panel.

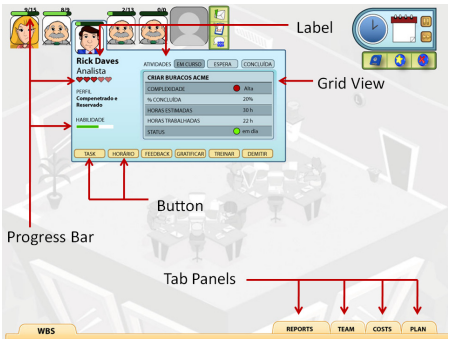


Figure 2: Interface components on a game screen panel

5.3 World View

Besides the user interface components, the Ingame Screen (Figure 3) shows an area portraying the scene: a software development office where the characters interact. This world view is too complex to be a simple interface component, so it is implemented as a class itself.

Figure 3: Ingame screen

The Ingame Screen is composed by several *Objects Views* which display each of the objects in the scene, like the characters, computers, furniture and the background. These are updated and drawn by the *World View*. The characters are implemented as

Character View, a specialization of *Objects View* with animation logic and several sprites.

5.4 Simulation

In order to create a believable software development environment, the characters of the game should have distinct skills and believable behaviors. In order to implement this feature, the characters are represented by *Synthetic Actors*: intelligent agents provided of emotions, personality and beliefs [Silva 2009]. These agents are focused on creating an illusion of life providing a good credibility to the player, rather than the usual intelligent agents approaches which aim at problem solving.

The logic behind these synthetic actors is controlled by the *AI Manager*, which constantly updates the state of the characters based on the player actions, the world's current state and the inference engine. The message exchange between *game* and *inference engine* is shown on Figure 4.

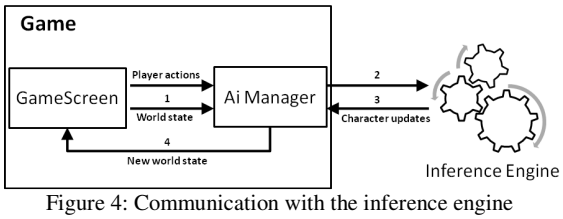


Figure 4: Communication with the inference engine

The game also needs to simulate the business rules of the software development industry, since teaching this knowledge is one of the game goals. For this purpose a *Project Manager* class was implemented, containing production rules based on project management theories [Pmbok 2008].

5.4 Resource Loading

The VTeam is a game directed to train project manager, and its sessions are usually conducted by an experienced instructor. Then, some customizations must be allowed. To provide this functionality, some features of the game can be changed by the user, via external tools such as the *Level Editor*. This tool is able to modify the XML file that describes game level, characters' personality and skills, etc.

6. Conclusion

Business simulation games have particular features that differ from the majority of conventional games. These features demand a customized architecture covering the requirements of this kind of game. Therefore, this paper provides to game independent developers an efficient and scalable architecture to develop business simulation games in XNA. This architecture was tested in a real game project, called VTeam.

However, the architecture should be validated in other projects, since the VTeam project does not cover all of its aspects. For instance, in this project there is only one view for each model, since there is a single view of the world view. In other words, the only visualization of the game's world is the room, where the characters work in the project. Thus, in a VTeam's extension another visualization could be available. For example, the project meeting room where the manager (player) could see the characters interactions and facial expressions.

Acknowledgements

First of all the authors would like to thank the VTeam development team and project coordinators for all the effort and support. Also thanks to the project partners and sponsors: FINEP, Jynx Playware, Valença & Associados, Qualiti, CNPq, FADE-UFPE and CIn-UFPE.

References

ESA, 2009. Essentials facts about the computer and video game industry. Entertainment Software Association. Available from: http://www.theesa.com/facts/pdfs/ESA_EF_2009.pdf [Accessed 19 June 2009].

LARAMÉE, F. D., 2002. Game Design Perspectives. Charles River Media.

SILVA, D. R D., 2009. Atores Sintéticos em Jogos Sérios: Uma Abordagem Baseada em Psicologia Organizacional. PhD thesis. Universidade Federal de Pernambuco.

VTEAM, 2009. Vteam. Available from: http://vteam.cin.ufpe.br. [Accessed 24 July 2009].

MOVIES, THE, 2008. Lionhead Studios – The Movies Online Story. Available from: http://www.lionhead.com/themovies/TMO.aspx [Accessed 22 July 2009].

MEIERS, S., 2006. Sid Meier's Railroads – Official Site. Available from: http://www.2kgames.com/railroads/railroads.html [Accessed 22 July 2009].

SMARTSIM, 2006. Smartsim – Serious Game com Atores Sintéticos. Available from: http://www.cin.ufpe.br/~smartsim/. [Accessed 27 July 2009].

XNA, 2009. XNA Creators Club [online]. Available from: http://creators.xna.com/en-US/ [Accessed 23 June 2009].

VIRTUAL UNIVERSITY, 2009. Virtual University Home Page. Available from: http://vu.org/. [Accessed 27 July 2009].

CAPTALISM, 1996. Capitalism for PC - Gamespot. Available from: http://www.gamespot.com/pc/strategy/capitalism/. [Accessed 27 July 2009].

PMBOK, 2008. A Guide to the Project Management Body of Knowledge. Project Management Institute.

GPU Octrees and Optimized Search

Daniel Madeira
Anselmo Montenegro
Esteban Clua
Computation Institute, UFF

Thomas Lewiner
Matmídia Laboratory, PUC-RIO

Abstract

Octree structures are widely used in graphic applications to accelerate the computation of geometric proximity relations. This data strucutre is fundamental for game engine architectures for a correct scene management and culling process. With the increasing power of graphics hardware, processing tasks are progressively ported of to those architectures. However, octrees are essentially hierarchical structures, and octree searches mainly sequential processes, which is not suited for GPU implementation. On one side, several strategies have been proposed for GPU octree data structure, most of them use hierarchical searches. On the other side, recent works introduced optimized searches which avoid hierarchical traversals.

In this work, we propose a GPU octree that allows for those optimized searches, which uses the GPU streaming to search for large of points at once. Moreover, we propose a parallelization of those optimized search to speed up the single point search.

Finally, the proposed structure takes advantage of the recent graphics hardware architectures to improve the GPU octree data structure.

Keywords:: Octree, GPU, Geometric Search, Hardware Acceleration

Author's Contact:

{dmadeira,anselmo,esteban}@ic.uff.br
lewiner@gmail.com

1 Introduction

A great amount of graphics algorithms rely on spatial proximity: collision detection, surface geometry approximation, particle-based fluid simulation, light ray-based rendering among many others. Brute-force search procedures to detect proximity relations typically induce a quadratic complexity, which is prohibitive for large data sets. Therefore, several optimizations have been proposed to accelerate those searches, most of them in a divide-and-conquer fashion: dividing the space into smaller blocks that can be processed independently. This division needs to be stored in memory, leading to a trade-off between memory consumption and search procedures acceleration.

Classical structures have been devised inside this trade-off: binary space partitions, k-d trees, octrees and multi-grids [Samet 1990]. Among them, the octree structure is certainly one the most widespread in image processing, geometric modeling, medical imaging, collision detection, point based rendering, isosurface visualization and volumetric rendering, among many other fields.

With the increasing power of graphics hardware (GPU), many graphics and non-graphics applications are ported to those parallelized stream-processing architectures, which is a very delicate but productive task [Buck et al. 2004; Zamith et al. 2008]. The ideal situation would be to handle all the application stage, and all of the geometry and rasterization stages on the GPU. However, several parts of the a typically application, such as user interaction or pure hierarchical traversals, are essentially sequential, requiring for a CPU implementation. This implies continuous CPU-GPU communications, which is often a bottleneck, since the data bus between CPU and GPU is limited.

This work proposes a GPU octree data structure that allows for optimized searches [Castro et al. 2008]. This structure is fully addressed in the GPU, reducing the CPU-GPU traffic.

2 Related Work

Recently, several works have proposed octree implementations on the GPU. Most of them represent the octree in the usual way of a general tree: each node has a reference for its eight children. The difference between them is how to reference the children of a node on the octree [Benson and Davis 2002; Lefebvre et al. 2005; Ziegler et al. 2007; Vasconcelos et al. 2008; Ajmera et al. 2008].

Some other representations have been proprosed, some of them replacing the explicit use of pointers to children by simple calculus of their location in an index table [Gargantini 1982; Glassner 1984; Warren and Salmon 1993; Lefebvre and Hoppe 2006; Bastos and Celes 2008].

Moreover, current GPUs offer much more efficient manipulation of data structures [Fatica and Luebke 2007; Nickolls et al. 2008] avoiding the restriction of using texture as mass memory, and this work introduces a simple way to efficiently take advantage of this evolution.

Usual searches in octrees proceeds in the divide-and-conquer manner: starting from the root node, the search decides at each node which child may contain a given location, and recurse on that child until reaching the leaves of the tree. This leads to an average logarithmic complexity of the search. Although sequential, this procedure is the base of most GPU octree proposals [Benson and Davis 2002; Lefebvre et al. 2005; Ziegler et al. 2007], or at least for the search of an isolated point [Vasconcelos et al. 2008; Bastos and Celes 2008]. Castro et al. [Castro et al. 2008] improved this search strategy for hash table representations of octrees. Instead of starting from the root, they begin searching at a given depth of the octree, and then traverse the octree up- or down-wards. A statistical optimization stated the initial depth to minimize the expected traversal steps, leading to an amortized constant time for the search. We use this strategy in this work.

Contributions

In this work, we propose a GPU octree search structure based on hash table that enjoys the new GPU architecture for the data structure and optimized search [Castro et al. 2008].

This optimized search strategy actually treats each search location and octree depth independently, which is optimal for the GPU parallel structure, and is the strength of our method. In particular, we can perform the search for a sequence of points in parallel, streaming the optimized search. We also introduce a parallel version of the optimized search, which allow to efficiently search for a single point at a time.

Our GPU octree search algorithm actually shows to be very fast. In the experiments reported in this work, our algorithm runs at least 3 times faster than the CPU algorithm, achieving, in some cases, an speed-up factor of 50.

3 Review of Octree Representations

An octree is a hierarchical data structure based on a recursive decomposition of a 3D region. Each node represents a cube in the region, and the root node represents the whole region. The cube of a non-leaf node is divided into 8 octants, thereby generating 8 children. In most applications, the data is stored in the leaves. In this section we present some common octree representations.

3.1 Pointer Octree

The most classic representation of octrees uses pointers, as a traditional tree. Usually, each node stores 8 pointers, one for each of its child, and some data. In leaf nodes, the pointers to the children are void, while in intermediate nodes, the data is void [Samet 1990]. A pointer from a child to his father can also be added, in order to facilitate upward traversal.

For octrees where each node is either a leaf or has exactly 8 children, it is possible to reduce the number of pointers by storing only a pointer to the first child and to the next sibling. However, this increases the traversal time.

3.2 Hashed Octree

It is possible to replace pointers by indexes. In that case, the references to a child node must be replaced by a calculus on the father's index, and the nodes must be stored in an index table.

Those structures are more compact that pointer octree, although depending on the pointer dereferencing time compared to the reference child index computation, it may be slower to access than pointer octrees.

However, they allow a direct access in constant time to any node of the octree, provided its index, while pointer octrees only allow direct access to the root.

000	001	010	011	100	101	110	111
10000	1	10010	10011	100	101	110	111
1111000	10001	1111010	1111011	1001100	1001101	1001110	1001111
	1111001			11100	11101	11110	11111

Figure 1: *Hash representation of the quadtree. The hash function uses a 3 bits key (top row) to group the tree nodes (bottom rows).*

The index can be generated *ad hoc* for a static octree in order to reduce the index table, as proposed on GPU through perfect hashing [Lefebvre and Hoppe 2006; Bastos and Celes 2008]. However, any significant change in the octree structure implies a complete rebuilding of the indexes, and the child references calculus are replaced by extra memory storage.

The index can also be systematically generated from the node geometrical position and/or by the node position in the octree hierarchy. This is the representation used in this work. A common choice for such index is Morton codes, reviewed at the next section. For usual octrees, those indexes are not consecutive, and thus the hash table must group them to avoid wasting too much memory. A common strategy is to group nodes that have the same last k bits of the Morton code, with $k = \lceil \log(n) \rceil + 1$, where n is the number of nodes (see Figure 1). In other words, the hash key h assigned to a morton code m is defined by $h(m) = m \mod k$.

3.3 Morton Index Generation

For spatial ordering of the nodes and to generate the indexes for the hashed octree, we use the Morton code. This method is efficient to generate unique index for each node, while offers god spatial locality and easy computation. Another advantage of Morton code is their hierarchical order, since it is possible to create a single index for each node, while preserving the tree hierarchy.

The index can be calculated from the tree hierarchy, recursively when traversing the tree. The root has index 1, and the index of each child node is the concatenation of its parent index with the direction of their octant, coded over 3 bits. The bottom-up traversal is also possible, as if to find the parent index we only have to truncate the last 3 bits of a child index.

The index can equivalently be computed from the geometric position of the node's cube and its size. Considering that the root's cube is a unit cube, and denoting by (x, y, z) the coordinates of the cube center and by 2^{-l} the cube side, i.e. the node is at depth l, we can generate the Morton code by:

$$1 x_l y_l z_l x_{l-1} y_{l-1} z_{l-1} x_{l-2} y_{l-2} z_{l-2} \ldots x_1 y_1 z_1 \,, \qquad (1)$$

where $x_l x_{l-1} \ldots x_1$ is the binary decomposition of $\lfloor 2^l x \rfloor$.

The generation of this index can be accelerated using integer dilation and contraction [Stocco and Schrack 1995].

4 Our GPU Octree Structure

We use hash table for the GPU representation of the Octree. This involves two specific design choices: the GPU memory used and the hash collision handling.

4.1 GPU Storage

Recent graphics hardware use CUDA architecture, which greatly improves the storage on GPU. CUDA stains for Computer Unified Architecture, and brings a new parading for memory distribution among the GPU. The texture memory can be addressed as a global memory for a GPU code, with linear and random addressing from the threads and blocks of each internal grid. This allow more efficient implementations of applications not related to the graphical pipeline and much more efficient octree implementations than presented in [Benson and Davis 2002], [Lefebvre et al. 2005] and [Ziegler et al. 2007].

4.2 Collision handling

A collision in the hash table occurs when two nodes have the same hash key. A simple option is to leave colliding nodes in the same entry of the hash table (open hashing). However, this requires either to allocate enough space in *all* the entries of the hash table to contain the maximal number of colliding nodes, which would require a static or controlled data and would waste a lot of memory, or to let a variable size structure in each entry of the hash table. This last option is commonly used in CPU implementation, since variable size container are easily implemented with pointers. However, this is much less efficient on GPU.

Another way to avoid collisions is to keep one of the colliding nodes in the position assigned by its hash key h, and to place the other ones at empty positions in the hash table (closed hashing). Those empty positions must be systematically chosen, by a collision function $c(h)$, to ensure that we can retrieve the other nodes!

Looking for a node n then resumes to looking at the position of its hash key. If the position is empty, then node n does not belong to the hash table. If not empty and if the node at the hash key position is different from n, then we look at position $h' = c(h)$. We repeat until n is found or a empty node is achieved. In this work, we use a linear colliding function, i.e. $c(h) = h + c_0$ for a fixed c_0.

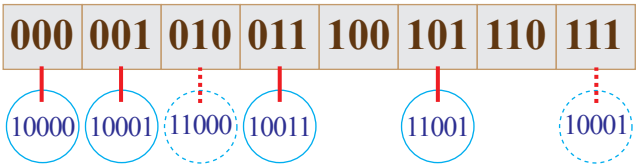


Figure 2: *Closed hashing: a colliding entry is sent to a new location, here its original location shifted by 2.*

5 Octree Search Algorithms

A direct search procedure in an octree returns the leaf whose cube contains a given position in space. In this section, we will recall usual algorithms for such search procedure, and introduce our procedure for our GPU octree.

5.1 CPU Algorithms

In pointer octrees, a search can only be done starting from the root node and traversing hierarchically the tree until the desired leaf is reached. This method has complexity of $\log_8(n)$, and $O(n)$ at worse case. The algorithm below shows a search in a pointer octree. In this method, the position and size of each traversed cube can be directly deduced from the recursion.

Algorithm 1: Classical search for point p.

```
start with the root r and a c a unit cube : n = r ;
while n is not a leaf do
    retrieve the child n' of n containg p ;
    set n = n' and c the octant of n' ;
end
return n;
```

Optimized searches offer a different access method. Since the leaves are the most distant nodes from the root node, it is better to start from a node closer to the desired leaves than from the root node.

However, to access a random node in the hashed octree, we need its Morton code, computable from position and depth. We know the position p from the search input, but the depth must be estimated. The optimized search [Castro et al. 2008] proposes to estimate this depth by the weighted median $\hat{l}$ of the expected depth, since it minimizes the number of traversal operations.

Algorithm 2: Optimized search for point p.

```
compute Morton code m_max of p at maximal depth;
compute code m of p at depth l=l̂ from m_max ;
access the node n corresponding to m in the hash table;
// upward traversal, in case n is below the leaf
while m does not belong in the hash table do
    decrease the depth l, removing 3 bits from m;
    access the parent of n in the hash table with m;
end
// downward traversal, in case n is not a leaf
while n exists in the hash table do
    increase the depth of m, adding 3 bits of m_max;
    access the child of n in the hash table with m;
end
return n as the last valid access to the table ;
```

5.2 GPU Algorithms

The main contribution of this work is to parallelize the optimal search algorithm, presented in the previous section, to port it to GPU architecture. To do so, we take advantage of the fact that the search for each point is independent and, that, in practical applications, e.g. collision, many queries are needed simultaneously.

We first describe how to parallelize the search for a single point p given g available threads. The main idea is to search at many levels in parallel. Since we know from the statistical optimization that the leaf is probably near level $\hat{l}$, we concentrate the threads around that level. In the CPU implementation, the upward and downward traversals test 1 level up or down. Here, to avoid duplicate efforts, the upward traversals are handled by the first half of the thread, skipping $\frac{g-1}{2}$ levels instead of 1, and the downward traversals. Although the CPU search can be as fast as the GPU search, the advantage of this method is fully maintain the octree in the GPU

Algorithm 3: GPU search for point p given g threads.

```
compute Morton code m_max of p at maximal depth;
for iter ∈ 1..iter_max do
    foreach thread t_i ∈ (0 ··· g−1) do
        if t_i ≤ (g−1)/2 then
            assign l_i = l̂ + t_i + iter · (g−1)/2 ;
        else
            assign l_i = l̂ + t_i − (iter + 2) · (g−1)/2 - 1 ;
        end
        compute code m of p at depth l_i from m_max ;
        access the node n corresp. to m in the hash;
        if n is a leaf then return n ;
    end
end
```

Observe that, in the CPU implementation, a leaf was detected testing for invalidate downward traversal. Here, we need to explicitly store for each node if it is a leaf or not. In practical applications, where the leaves are the only nodes containing data, this does not induce any memory overhead.

Now, computing the search for many points at the same resumes to divide the number of available threads by the number of points to search for, and use the above algorithm in parallel for each point.

6 Experiments and Results

We implemented the above algorithms inside the CUDA architecture, using the hashed octree with closed hashing. To handle the colisions, we used the linear colliding function. Although there could be more optimized methods, we chose it for its simplicity.

In our experiments, we used five models, at different resolutions. The results were obtained on a Core 2 Duo T9400 CPU, running at 2.53GHz, with a GeForce 9600M GT, with 32 processors.

We first tested our approach by searching for random points, and compare the timings between the CPU and GPU implementations. We searched simultaneously for 10 to 300 points.

The absolute timing comparisons are reported on Table 3, our algorithm achieved an execution time at least 3 times faster than CPU algorithm for a small number of searches. When searching for 300 points, our algorithm runs in average 45 times faster.

On the detailed results reported on Tables 1 (CPU) and 2 (GPU), we clearly see that the increase of the number of search points generates a linear increase of the execution time with a high proportion, while the results on the GPU shows a clear benefit of our parallelization.

In Figure 3 we can observed that the execution time of our algorithm grows slowly, in a linear pattern.

Table 1: *Total execution time (in seconds) for the CPU algorithm.*

# points	**10**	**50**	**100**	**200**	**300**
Ant	1	2	5	10	14
Armadillo	1	4	8	17	25
Bunny	1	3	7	14	21
Drill vripped	1	2	4	6	10
Drill zip	1	3	7	15	21

We further test for the influence of the hash table size. While in the first test, the octrees of all models and all implementations were stored in a hash table with the same size, in this second test, we varied the size of the hash table. In the results are presented in Table 4, we check that the influence of the size on our GPU algorithm is little, while keeping it big enough to contain the whole octree.

Table 2: *Total execution time (in seconds) for our GPU algorithm.*

# points	10	50	100	200	300
Ant	0.27	0.28	0.36	0.39	0.43
Armadillo	0.29	0.29	0.38	0.41	0.44
Bunny	0.28	0.35	0.37	0.4	0.37
Drill vripped	0.28	0.35	0.37	0.34	0.37
Drill zip	0.28	0.28	0.21	0.37	0.38

Table 3: *Speed-up of our algorithm for the test cases. Our algorithm runs at least 3 times faster than the CPU algorithm.*

	10	50	100	200	300
Ant	3,70	7,14	13,89	25,64	32,56
Armadillo	3,45	13,79	21,05	41,46	56,82
Bunny	3,57	8,57	18,92	35,00	56,76
Drill vripped	3,57	5,71	10,81	17,65	27,03
Drill zip	3,57	10,71	33,33	40,54	55,26

7 Conclusion

This work presented a new GPU approach for searching elements in octrees, based on hash table representation and optimized search. The solution is based on the fact that, in such contexts, each octree level is independent from the others. We proposed a parallelization of the search algorithm, which lead to a totally streamed implementation. This method combines the memory efficiency of the hashed octree with a significantly smaller execution time thanks to the structure of recent graphics hardware.

Many diferent applications can benefit from this speed-up, since it can be implemented to any scene structure that supports octree representation.

As future works, more efficient hash methods can be analyzed and used with the presented strategy. It is also important to validade the proposed approach for complete applications, such as real time 3D collision detection.

Also with the 2 searchs presented, we can now implement the octree generation on the GPU.

References

AJMERA, P., GORADIA, R., CHANDRAN, S., AND ALURU, S. 2008. Fast, parallel, GPU-based space filling curves and octrees.

BASTOS, T., AND CELES, W. 2008. GPU-accelerated Adaptively Sampled Distance Fields. In *IEEE International Conference on Shape Modeling and Applications, 2008. SMI 2008*, 171–178.

BENSON, D., AND DAVIS, J. 2002. Octree textures. In *Siggraph*, vol. 21, 785–790.

BUCK, I., FOLEY, T., HORN, D., SUGERMAN, J., FATAHALIAN, K., HOUSTON, M., AND HANRAHAN, P. 2004. Brook for GPUs: stream computing on graphics hardware. In *Siggraph*, 777–786.

CASTRO, R., LEWINER, T., LOPES, H., TAVARES, G., AND BORDIGNON, A. L. 2008. Statistical optimization of octree searches. *Computer Graphics Forum 27*, 1557–1566.

FATICA, M., AND LUEBKE, D., 2007. High performance computing with CUDA. Supercomputing 2007 tutorial. In Supercomputing 2007 tutorial notes, November.

GARGANTINI, I. 1982. Linear octrees for fast processing of three-dimensional objects. *Computer Graphics and Image Processing 4*, 20, 365–374.

GLASSNER, A. 1984. Space subdivision for fast ray tracing. *Computer Graphics & Applications 4*, 10, 15–22.

LEFEBVRE, S., AND HOPPE, H. 2006. Perfect spatial hashing. In *Siggraph*, 579–588.

Table 4: *Execution times (in seconds) for the Ant model, with different sizes for the hash table.*

	Size of the hash table				
	2,000,000	**1,000,000**	**500,000**	**250,000**	**100,000**
CPU	5 s	2 s	1 s	1 s	1 s
GPU	0.33 s	0.29 s	0.28 s	0.28 s	0.27 s

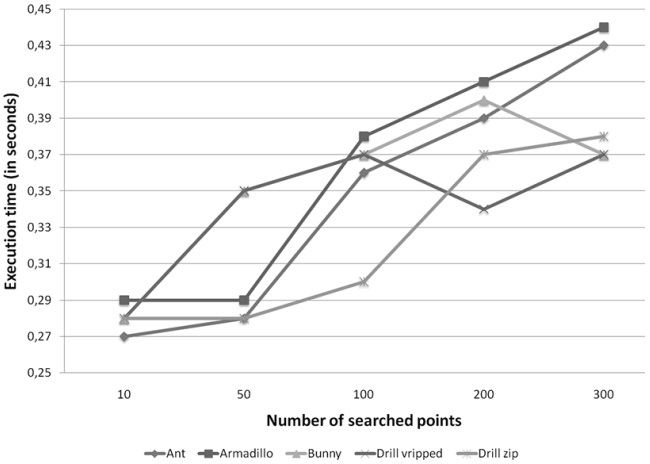


Figure 3: *Execution time of the GPU results of Table 2. The graph shows the linear complexity of our algorithm.*

LEFEBVRE, S., HORNUS, S., AND NEYRET, F. 2005. Octree textures on the GPU. In *GPU Gems 2 - Programming Techniques for High-Performance Graphics and General-Purpose Computation*. Addison Wesley, ch. 37, 595–613.

NICKOLLS, J., BUCK, I., GARLAND, M., AND SKADRON, K. 2008. Scalable parallel programming with cuda. *Queue 6*, 2, 40–53.

SAMET, H. 1990. *The design and analysis of spatial data structures*. Addison-Wesley.

STOCCO, L., AND SCHRACK, G. 1995. Integer dilation and contraction for quadtrees and octrees. In *IEEE Pacific Rim Conference on Communications, Computers, and Signal Processing, 1995. Proceedings*, 426–428.

VASCONCELOS, C., SÁ, A., CARVALHO, P., AND GATTASS, M. 2008. Quadn4tree: A gpu-friendly quadtree leaves neighborhood structure. In *CGI: Computer Graphics International*. 1101.

WARREN, M. S., AND SALMON, J. K. 1993. A parallel hashed octree n-body algorithm. *Supercomputing, IEEE*, 12–21.

ZAMITH, M., CLUA, E. W. G., PAGLIOSA, P., CONCI, A., VALENTE, L., FEIJO, B., LEAL, R., AND MONTENEGRO, A. 2008. The GPU used as a math co-processor in real time applications. *Journal of Computer in Entertainment: CIE 6*, 1–19.

ZIEGLER, G., DIMITROV, R., THEOBALT, C., AND SEIDEL, H. 2007. Real-time quadtree analysis using HistoPyramids. In *IS&T and SPIE Conference on Electronic Imaging*, vol. 6496. 0L.

Jogo de Nave como Ferramenta para Auxílio ao Ensino de Astronomia

Thiago Luiz da Silva, Benjamin Grando Moreira

Universidade do Vale do Itajaí (UNIVALI),
Centro de Ciências Tecnológicas da Terra e do Mar, Itajaí, SC, Brasil

Resumo

Este artigo descreve o desenvolvimento de um jogo de nave para o ensino de astronomia para crianças de 5º série bem como a evolução e a utilização da informática e de jogos na educação. Neste artigo pode-se encontrar o detalhamento da ferramenta proposta, informações sobre desenvolvimento de jogos, análise de ferramentas similares jogos na educação e as informações de astronomia que serão disponibilizadas no jogo.

Palavras-chave: Jogos na Educação, Astronomia, XNA.

Contato:
{thiago.silva,benjamin}@univali.br

1. Introdução

O ensino por meio de jogos pode se tornar uma ajuda bem vinda ao ensino devido a interatividade que o mesmo proporciona. Em alguns casos, o ensino textual, ou até vídeos de documentários podem acabar confundindo os alunos. Quando as aulas são muito teóricas, o ensino pode se tornar tedioso dificultando o aprendizado do aluno. Nesse sentido, segundo Soares [2001], pode-se oferecer o ensino por meio de jogos, criando uma situação simulada do mundo real, tornando o aluno mais questionador e impulsivo a aprendizagem do conteúdo.

O uso de jogos eletrônicos como ferramenta de auxílio no ensino pode ser de boa utilidade, pois, criar situações interessantes para ensinar, faz com que o aluno associe o aprendizado ao prazer [Lima 2005].

Assim, um jogo eletrônico bem desenvolvido pode não só facilitar o ensino do professor, como facilitar a aprendizagem dos alunos.

O jogo aqui sendo descrito é um jogo de nave espacial que abordará o conteúdo de astronomia ministrado para crianças 5º série onde são apresentados os conceitos sobre os planetas, como seus nomes, características e ordem de órbitas.

2. Jogos na Educação

O jogo como meio de ensino sempre fez parte da sociedade, sendo ele mais um bem comum entre homens e animais. Sendo assim, jogos são utilizados como objeto de aprendizagem e treinamento á muitos séculos [Huizinga 2000 *apud* Montero 2007]. Jogar é algo que se faz por vontade, sem nenhuma obrigação, e é isso que faz com que o jogo seja uma maneira interessante de se utilizar para a aprendizagem.

Com a popularização do computador, um grande conjunto de áreas sofreu mudanças para se adaptar ao seu uso. Com a evolução de métodos de ensino houve a mudança do pensamento em que o professor deixa de ser um simples repetidor e passa a ser um facilitador do saber. Com estas mudanças, ao perceber que o processo de recepção de conhecimento está vinculado ao processo de formação do indivíduo como ser social, pensante e crítico, a necessidade de novos métodos de ensino foi percebida.

Desse modo, um jogo educacional não pode ser reduzido a um simples questionário, ou alguns textos com interações básicas que limitam a atuação do aluno como um ser pensante. Como menciona Aranha [2006], há uma grande ausência de conhecimento sobre o papel do computador no ensino por parte dos profissionais oriundos da área de informática. Neste caso "é necessário utilizar os jogos eletrônicos não como banco de dados (análise errônea de uma mentalidade habituada com a lógica impressa), mas como ferramenta de motivação" [Aranha 2006].

A criação de jogos eletrônicos como ferramenta de ensino é complicada e faz necessário pensar em todos os detalhes. O jogo não pode ser altamente divertido, cheio de recursos se estes dispensarem a atenção do aluno ao conteúdo em que o jogo foi programado para passar. Em contra partida, não se pode fazer um jogo em que não há desafios, frustrando o jogador e toda a equipe que trabalhou no projeto. Os recursos do jogo devem ser calibrados para que a motivação do jogador seja garantida e que o conteúdo ensinado seja absorvido quase de maneira imperceptível. Princípios como imersão e interatividade são os pontos chaves para motivar os alunos a jogarem um jogo educacional.

3. Jogos Similares

Com relação a trabalhos similares, foram analisados os jogos [Gui jogos 2008; Terra 2009; Klick Educação 2006; MEC 2004; e XNA Creators Club 2008], todos com relação a questões espaciais. Conforme os estudos

realizados com as ferramentas acima foram levantadas as seguintes características:

- **Informações Básicas do Sistema Solar:** informa se o jogo contempla as informações básicas de astronomia ensinados na 5º série, como sol, planetas, nomes e ordem de órbitas;
- **Informações Avançadas do Sistema Solar:** informa se o jogo contempla informações avançadas de astronomia como dimensão, dias de translação e rotação, atmosfera, entre outros;
- **Interatividade:** quantidades de movimentos que o jogo possui e qual o grau de desafio ele proporciona;
- **Gráfico:** visa observar as texturas utilizadas no jogo e os efeitos de tela para cada movimento efetuado;
- **Som:** visa observar se o jogo possui recurso de áudio juntamente com sua qualidade;
- **Diversão:** neste caso, a diversão é o conjunto das características anteriores; e

Analisando os jogos foi possível perceber que jogos educacionais, embora tenham usado som e interatividade, não possuem características de um bom jogo (como interatividade e diversão) cumprindo bem apenas a exibição do conteúdo. Já os jogos criados para a diversão suprem esta carência, mas tratam superficialmente a questão de conteúdo. Desta forma, com a união dos gráficos dos jogos não educacionais com as informações dos educacionais é esperado um jogo excitante de se jogar.

4. Metodologia

Os jogos atraem a atenção dos alunos, tornando o ensino muito mais fácil e divertido [Lima 2005]. Além de prender a atenção dos alunos, o ensino por meio de jogos pode utilizar múltiplas maneiras de ensinar um mesmo conteúdo, adequando o sistema para cada aluno.

Com os últimos avanços científicos na área espacial, como descoberta de novos planetas, reclassificação dos planetas e novas tecnologias, a maiorias dos livros de geografia estão desatualizados e terão que sofrer alterações. Neste caso, um jogo neste tema pode ser bem aproveitado. Este jogo pode não só ensinar o básico sobre este tema como pode facilmente incorporar informações e situações extra-classe aumentando o conhecimento dos alunos.

Para elaboração do jogo, um estudo sobre os conteúdos de geografia da 5º série, foi realizado e identificado os assuntos tratados. Segundo alguns livros de geografia da 5º série, Luci [1996] e Valle [2005], essa série é o momento em que as crianças começam a aprender sobre o sistema solar, devendo ser apresentada algumas informações relevantes e de obrigatoriedade sobre o tema: conceitos básicos de astronomia, como Big Bang, corpos celestes, órbita, estrelas, galáxias, universo, planetas, satélites naturais, asteróides e meteoritos e medida de anos-luz.

4.1. Proposta

Este trabalho consiste em oferecer ao aluno um jogo de nave que se passa no sistema solar. Segundo opiniões recolhidas de professores, será oferecido ao aluno conceitos básicos de astronomia, como os planetas de nosso sistema solar com suas respectivas características (dimensão, ordem de órbita, nome, satélites, atmosfera, entre outras características básicas), e conceitos básicos de ciências, trazendo a composição do solo e gases dos planetas. Estas características serão visualizadas conforme o aluno for realizando as missões disponíveis no jogo. Matemática também será necessária para cálculos de distância e combustível da nave.

A intenção é criar diversas missões que garantam que o aluno visite todos os planetas do sistema e verifique suas características. Para satisfazer a necessidade básica de um jogo, como desafio, respeito e socialização, alguns detalhes precisam ser trabalhados, dentre eles um ranking de medalhas de premiação. Conforme o tempo que o aluno realize uma determinada missão, ele receberá uma medalha diferenciada. Existirão patentes e um ranking de pontuação total, assim gerando alguma competição.

Também será exibida uma tabela onde o aluno deverá preencher as opções completando a tabela de planetas. O jogo termina quando todas as missões forem finalizadas e a tabela estiver completa. O jogo também poderá ser jogado em modo multijogador, onde o professor vai possuir uma tela de controle de vôo, verificando todo o posicionamento dos alunos e dos planetas. Os alunos poderão interagir entre si para a conclusão de seus objetivos ou de um objetivo em comum informado pelo professor.

Este projeto é a continuidade de um jogo de nave idealizado na matéria de Informática na Educação que foi desenvolvido com a ferramenta pascal. Atualmente, devido a popularização dos jogos e a disponibilização de bibliotecas gratuitas para sua criação, o jogo está sendo refeito em Visual C# e XNA (Biblioteca gráfica Microsoft utilizada para criação de jogos para microcomputadores e Xbox 360). Com estas mudanças, busca-se facilitar o desenvolvimento, minimizar a quantidade de lags (problema de travamento que reproduz a animação de forma lenta) existentes na primeira versão e melhorar a qualidade gráfica e de movimento do jogo (Figura 1).

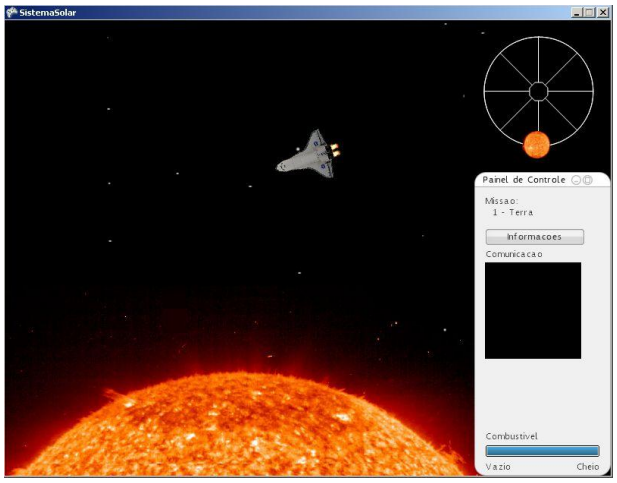


Figura 1. Protótipo da nova versão em Visual C# e XNA

A ferramenta XNA Game Studio é uma Framework criada pela Microsoft para apoiar desenvolvedores que programam por diversão e acadêmicos a desenvolver jogos com mais qualidade e facilidade para microcomputadores com sistema operacional Windows, para o aparelho Zunes e para o videogame Xbox 360. A versão atual da ferramenta XNA está na 3.0 e está integrada com o Microsoft Visual Studio C#. Com ela, o desenvolvedor pode cruzar facilmente as características de seu jogo para rodar nas três plataformas disponíveis anteriormente. A nova versão também abre o acesso à conexão da Xbox Live, um ambiente de rede que interliga os jogadores de Xbox 360 em todo o mundo. Com esta conexão, o jogador pode disponibilizar seu jogo para estas pessoas, alem de possibilitar o desenvolvimento de jogos multiusuários. A ferramenta possui um site próprio nomeado de XNA Creator Club. Neste site é possível encontrar diversos tutoriais em vídeos, ajuda em texto, fórum e códigos fontes de exemplos para a demonstração da maioria dos recursos básicos que a ferramenta oferece [Microsoft 2009].

5. Roteiros

Antes de iniciar as missões como um astronauta, o jogador devera ajudar a criar o Big Bang. Com isso, o aluno poderá ver como foi a formação do universo. Inicialmente, irá existir um fundo preto com apenas um ponto central luminoso. Uma mensagem será exibida ao usuário informando que o mesmo deverá ajudar na criação do Big Bang expandindo este ponto luminoso.

A partir de certo momento, a animação ocorrerá sem a necessidade de interação. A animação exibirá a expansão do universo fazendo com que várias estrelas se distanciem do centro. Paralelamente a este evento, as estrelas começarão a se unir formando o desenho tradicional das galáxias.

Após as galáxias estarem dispersas pela tela, uma aproximação exibirá a Via Láctea. Após mostrar informações, uma nova aproximação irá ocorrer no ponto onde se localiza o sistema solar. Em seguida, será realizada uma animação como se o aluno já fosse o astronauta e estivesse na base momentos antes de ser lançado ao espaço apenas recebendo instruções. Estas instruções remeterão a uma missão. Atualmente, 10 missões foram elaboradas, mas apenas 4 estão melhor definidas.

5.1 Primeira Missão

A primeira missão tem por objetivo recolher três satélites e encaminhá-los a estação espacial. Com relação ao ensino, ela visa conscientizar os alunos sobre o lixo espacial e alimentar seu conhecimento sobre a estação espacial internacional.

Essa missão consiste em remover três satélites inutilizados para a estação espacial para que sejam reciclados. Sempre que a frente da nave estiver apontada para o satélite e uma determinada tecla for pressionada, uma corda será lançada fixando uma garra no satélite. Ao se aproximar da estação espacial pela primeira vez, uma mensagem será exibida dando detalhes e trazendo as informações sobre a estação espacial internacional. Sempre que um satélite estiver ligado a nave e ela passar perto da estação espacial, o satélite será removido e contabilizado como ponto ao usuário. A Figura 2 ilustra a versão inicial dessa missão. Ainda não foram colocados os planetas em suas respectivas proporções e distâncias.

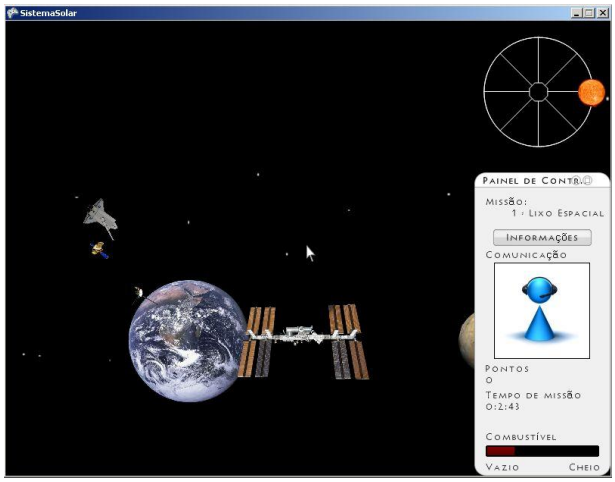


Figura 2. Protótipo da primeira missão

5.2 Segunda missão

Na segunda missão o objetivo é viajar até Marte para fazer estudos sobre a possibilidade de manter ali uma estação de reabastecimento e visa ensinar ao aluno mais sobre o planeta Marte. O jogador deve viajar até marte para verificar o detalhamento de seu solo e sua atmosfera. Uma mensagem será exibida informando para o usuário qual é o combustível utilizado no foguete e quais as características necessárias que um planeta deve ter para que seja possível montar um centro de reabastecimento. Assim que o jogador confirmar a existência de gelo em marte, sua missão estará completa.

5.3 Terceira missão

Essa missão tem por objetivo buscar uma nave sem combustível próximo a Vênus e rebocá-la até a estação espacial. Com relação ao conteúdo de ensino, essa missão visa conhecer mais sobre o planeta Vênus. Uma mensagem será exibida ao aluno informando que uma situação crítica ocorreu e que uma nave ficou sem combustível próximo a um planeta aparentemente próximo do sol. A nave foi pega por uma chuva de meteoros e a peça responsável por carregar a bateria pela luz solar ficou perdida próxima ao planeta. Com isso, seu localizador está desligado sem saber ao certo onde estão.

5.4 Quarta missão

A Missão 4 tem por objetivo verificar temperatura solar e com isso apresentar mais detalhes do Sol e de outras estrelas. Uma mensagem será exibida ao aluno informando que para a continuidade de uma pesquisa, é necessário medir a temperatura solar com mais precisão. Para isso, o jogador deverá se dirigir ao sol. Ao chegar lá, o jogo deve informar ao usuário informações básicas sobre as estrelas. Porque ele vê pontinhos brilhantes no céu e porque isso só acontece a noite. Deve também informar a temperatura média do sol na superfície para que o aluno possa finalizar sua missão. Para finalizar esta tarefa, o aluno devera enviar um sinal de rádio informando qual a temperatura o termômetro especial calculou.

6. Considerações finais

Ao realizar as análises dos jogos similares, foi necessário levantar informações de jogos educacionais sobre o sistema solar e de jogos de naves não educacionais. Essa análise foi necessária para que o projeto fosse enriquecido tanto com informações quanto com gráficos e jogabilidade de jogos mais avançados. O levantamento encontrou alguns dados interessantes, principalmente quanto o grau de divertimento dos jogos educacionais. A maioria se mostrou por caracterizar apenas por jogos de perguntas.

7. Referências

ARANHA, G., 2006. *Jogos Eletrônicos Como um Conceito Chave para o Desenvolvimento de* Aplicações Emissivas e Interativas para o Aprendizado. Ciências e Cognição.

CAMARGO, Nelson José de, 2005. Atlas Geográfico Novo Milênio, São Paulo – Editora Didática Brasil.

GUI JOGOS. 3D Solar System

<http://www.guijogos.com/jogar/792/3D_Solar_System/> [Acesso em 20 Mar. 2009]

KLICK EDUCAÇÃO. Geografia <http://ig.klickeducacao.com.br/2006/materia/16/display/0,5912,PPR-16-42-2216-,00.html> [Acesso em 03 Abr. 2009]

LIMA, Elvira Souza, 2005. *Revista Nova Escola edição 179* – janeiro/fevereiro.

LUCCI, Elian Alabi, 1996. *Geografia, Homem e Espaço* – Volume 1, Edição 10 – Editora Saraiva,.

MEC. *Banco Internacional de Objetos Educacionais* <http://objetoseducacionais2.mec.gov.br/handle/mec/2990> [Acesso em 27 Fev. 2009]

MICROSOFT. *XNA Developer Center* <http://msdn.microsoft.com/pt-br/xna/default(en-us).aspx> [Acesso em 15 abr. 2009]

PERUCIA, Alexandre S., 2007. *Desenvolvimento de Jogos Eletrônicos Teoria e Prática* – 2º Edição, Novatec.

SOARES, Wilson Marcos, 2001. *Revista Nova Escola edição 145* – setembro.

TERRA. *Jogos* <http://www.terra.com.br/criancas/jogos_planeta.htm> [Acesso em 20 Mar. 2009]

VALLE, Cecília, 2005. *Coleção Ciências 5º Série* – Edição única, Positivo.

XNA *Creators Club* <http://creators.xna.com/en-US/education/catalog/?devarea=19> [Acesso em 20 Mar. 2009]

MCommP2P: Um Middleware P2P para Jogos em Rede

Roberio Kielmann Daniel G. Costa João B. Rocha-Junior

Departamento de Tecnologia
Universidade Estadual de Feira de Santana, Bahia, Brasil

Abstract

Nowadays, most of the electronic games communicate using the client-server paradigm, needing specific centralized machines acting as servers. P2P paradigm brings advantages in load distribution, dispensing the use of servers. In such context, a P2P middleware for game developing is created. Such middleware addresses the communications features related with networked games, such as state control and secure data exchange, bringing a new flexible and pluggable solution for the development of games.

Keywords: P2P communication, networked games, middleware.

Resumo

Atualmente, a maior parte dos jogos em rede se comunica utilizando o paradigma cliente-servidor, demandando processamento centralizado em máquinas específicas. O uso do paradigma P2P traz vantagens na distribuição de carga, dispensando o uso de servidores. Nesse contexto, foi desenvolvido um middleware de comunicação para jogos utilizando este paradigma. Esse middleware realiza diversas tarefas necessárias aos jogos em rede, como controle de consistência de estados e comunicação segura, se apresentado como uma solução flexível e fácil de usar no desenvolvimento de jogos.

Palavras-chave: Comunicação P2P, jogos em rede, middleware.

Contato dos autores:
{roberio,daniel,joao}@ecomp.uefs.br

1. Introdução

A indústria nacional de jogos ainda está em um ciclo inicial de amadurecimento, resultando em uma grande distância entre os produtos nacionais e estrangeiros. Isto faz com que motores de jogos (*game engines*), ferramentas RAD (Rapid Application Development) e middlewares especializados tornem-se essenciais para o desenvolvimento de jogos, aumentando seu realismo, jogabilidade e competitividade junto ao mercado [Bittencourt e Osório 2006].

Os jogos eletrônicos em rede destacam-se como uma forte tendência dentro do mercado de entretenimento, apesar de possuírem potencialmente maior custo e complexidade de desenvolvimento. Isto ocorre devido, entre outras coisas, à popularização de tecnologias como a *web*, a difusão do uso de computadores e o barateamento do acesso à Internet.

Atualmente, a maior parte dos jogos que fornecem suporte a comunicação em rede utilizam o paradigma cliente-servidor, devido, entre outros fatores, à menor complexidade e custo de desenvolvimento em relação ao paradigma P2P (peer to peer) [Azambuja 2006]. Contudo, comunicações baseadas no paradigma cliente-servidor tornam necessária a utilização de máquinas específicas e geralmente com maior largura de banda para atuarem como servidor devido à centralização de informações. Isso pode ser tornar um problema para pequenas empresas desenvolvedoras de jogos, devido ao investimento e manutenção desta estrutura, e jogadores, que precisarão definir máquinas para atuarem como servidor [Jábali 2004].

O uso do paradigma P2P no desenvolvimento de jogos *multiplayer*, apesar de ter sua implementação potencialmente mais complexa devido a questões de consistência de estados e monitoramento de fraudes, gera vantagens na distribuição de carga, visto que todas as máquinas dividem as tarefas do servidor. Este paradigma, portanto, dispensa o uso de uma máquina específica centralizando os dados.

Nesse contexto, foi desenvolvido um middleware de comunicação para jogos utilizando este paradigma, o MCommP2P, visando simplificar e reduzir tempo e custos de desenvolvimento de jogos por pequenas empresas e grupos de estudo na área. Esse middleware trata questões específicas dos jogos em rede, como consistência de estados, segurança dos dados que trafegam na rede, busca por sessões de jogos a iniciar, entre outros. A idéia é que esse middleware possa ser utilizado para facilitar o desenvolvimento de jogos em rede, podendo ser utilizado diretamente em jogos desse tipo ou ser incorporado como um módulo em motores de jogos que possuam uma arquitetura que favoreça essa modularização.

Este artigo está organizado da seguinte forma. Na segunda seção é apresentado o desenvolvimento de jogos em rede. Em seguida, os detalhes de especificação e implementação do middleware MCommP2P são apresentados. Além disso, são apresentados nessa seção os testes e resultados do

middleware, mostrando como foi feita sua integração com os protótipos de jogos desenvolvidos, além da discussão de resultados. Por ultimo encontram-se as considerações finais e referências do trabalho.

2. Desenvolvendo jogos em rede

Há algumas décadas, os jogos que possibilitavam a comunicação em rede destacavam-se dos demais, visto que tinham esta opção como um diferencial. Atualmente, oferecer a possibilidade de diversos jogadores participarem de uma mesma partida em um jogo sem que estejam reunidos presencialmente ou interconectados no mesmo dispositivo é um requisito comum [Bittencourt e Osório 2006]. Dentro da grande quantidade de jogos em rede disponíveis no mercado, destacam-se quatro grandes grupos: First Person Shooters (FPS), Real Time Strategy (RTS), Massively Multiplayer Online Game (MMOG) e Non Real Time (NRT) [Anibolete 2006].

Alguns desafios são comuns a todos os jogos em rede, devendo haver um maior esforço para tentar resolve-los ou minimizá-los durante o desenvolvimento de jogos deste tipo. Entre os desafios mais comuns nessa área estão a consistência de estados, o controle de fraudes e a segurança na comunicação.

Manter a consistência de estados significa garantir que todos os jogadores possuem o mesmo estado do jogo e que concordam com cada acontecimento desse jogo. Esta funcionalidade é considerada a mais complexa, visto que se torna difícil garantir a consistência de estado sem causar perdas de desempenho da aplicação. Para a sincronização do estado entre todos os computadores envolvidos na partida, diversos algoritmos distribuídos otimistas e pessimistas são propostos. Os algoritmos distribuídos pessimistas são projetados para evitar a ocorrência de inconsistências, não estando preparados para corrigi-las, enquanto que os otimistas permitem a ocorrência de inconsistências, realizando a correção assim que identificadas [Anibolete 2006]. Alguns dos principais algoritmos distribuídos são o stop-and-wait, o bucket synchronization e o Trailing State Synchronization, sendo o primeiro pessimista e os demais otimistas.

Para dificultar a ação de jogadores mal intencionados é necessária a utilização de mecanismos de criptografia dos pacotes transmitidos na rede, já que capturar pacotes para análise e geração de outros, que possam dar vantagens ao jogador, é considerada uma prática comum [Kozovits 2003]. Os algoritmos simétricos, que utilizam a mesma chave para criptografar e decriptografar, são considerados a melhor opção de criptografia de pacotes, pois possuem uma menor complexidade temporal quando comparados aos assimétricos. Algoritmos assimétricos têm sua complexidade aumentada por utilizarem um par de chaves, uma pública e outra privada, usadas na criptografia e decriptografia, respectivamente [Hinz 2000]. Neste contexto, os algoritmos assimétricos podem ser usados para garantir o compartilhamento da chave simétrica de forma segura.

Em jogos com número elevado de jogadores, não se pode assumir que todos estejam agindo corretamente, visto que é comum existirem jogadores que tentam obter vantagens irregulares durante o jogo. Sendo assim, torna-se necessário o monitoramento e combate às fraudes que ocorrem durante o jogo, visando minimizar o número de vitórias fraudulentas e restringir ou desestimular a quantidade de tentativas de violação [Baughman e Levine 2007]. Uma das alternativas nesse sentido é guardar um arquivo de *log* contendo a data e a hora de inicio e fim da seção, bem como ocorrências suspeitas. A partir da análise estatística destes dados é possível identificar o nível de confiabilidade de algum jogador, que caso seja baixo pode ter suas atividades restringidas ou bloqueadas. Em casos de ocorrências de atividades suspeitas, podem ser enviadas mensagens de notificação ao jogador de forma não determinística para que seja despertada a dúvida se está ou não sendo monitorado [Baughman e Levine 2007]. Todas essas funcionalidades, que devem ser levadas em consideração no desenvolvimento de jogos em rede, estão presentes no middleware MCommP2P.

3. O Middleware MCommP2P

Visando facilitar o desenvolvimento de jogos em rede, foi criado um middleware de comunicação que já implementa diversos serviços relacionados aos jogos desse tipo. Esses serviços são disponibilizados seguindo o paradigma P2P, não sendo obrigatório, nesse caso, o uso de servidores centralizando o controle das informações. A idéia foi criar um recurso que facilite e acelere o desenvolvimento de jogos, apoiando diretamente pequenos desenvolvedores.

O middleware desenvolvido oferece serviços relacionados à comunicação de jogos em rede, endereçando problemas como consistência de estados, controle de sessão, criptografia de pacotes e monitoramento de fraudes. Desta forma, estes serviços podem ser abstraídos pelo usuário, que poderá escolher quais funcionalidades oferecidas serão utilizadas, tornando o desenvolvimento de jogos em rede mais simples. Todos esses serviços podem ser configurados através de um arquivo de definições em XML.

Outra característica do software desenvolvido está relacionada ao paradigma de comunicação P2P, utilizado durante a dinâmica do jogo. Este paradigma foi escolhido para que os jogos desenvolvidos pudessem ser os mais independentes possíveis de um controlador central, reduzindo assim os custos. Algumas das funcionalidades oferecidas ainda possuem dependência com um servidor, como a autenticação de usuários, devido à dificuldade de persistência de dados em redes P2P. Contudo,

características como estas podem ser desabilitadas pelo desenvolvedor, caso não sejam necessárias.

A consistência de estados foi implementada no middleware seguindo o algoritmo stop-and-wait. Este algoritmo limita-se a bloquear o envio de mensagens até que todos tenham enviado no turno anterior, funcionando como uma barreira cíclica. A quantidade de mensagens que deve ser recebida para que o envio esteja liberado é definido após o início da sessão.

Para evitar que esta forma de sincronização possa atrapalhar a integração com algum jogo, e permitir aos desenvolvedores a opção de realizar sua própria sincronização através de outros algoritmos, é oferecida a opção de desativação da barreira no arquivo de configuração do middleware. Com a barreira desativada, as mensagens são enviadas imediatamente, devendo a consistência de estado ser garantida, nesse caso, pelo desenvolvedor do jogo. Após passar pela barreira, a mensagem é enviada a cada um dos *hosts* de uma sessão, sendo para isto serializada, criptografada, quando a criptografia está ativa, e enviada através do canal de comunicação entre os *hosts*. Os algoritmos de criptografia utilizados na implementação inicial do middleware foram o RSA e o AES.

Para monitorar fraudes, o middleware MCommP2P valida os pacotes recebidos através da captura de erros durante a deserialização dos objetos transmitidos. A ocorrência de erros neste processo indica a alteração de conteúdo do pacote, sendo a suspeita encaminhada ao módulo Monitor (pertencente ao middleware). Ao receber uma notificação, o módulo Monitor pode realizar dois procedimentos: notificar o jogador suspeito, sendo isto feito de forma não determinística para despertar a dúvida sobre seu monitoramento, ou notificar um servidor informado no arquivo de configuração, para que este possa analisar estatisticamente as ocorrências. A figura 1 apresenta o digrama de classes do middleware.

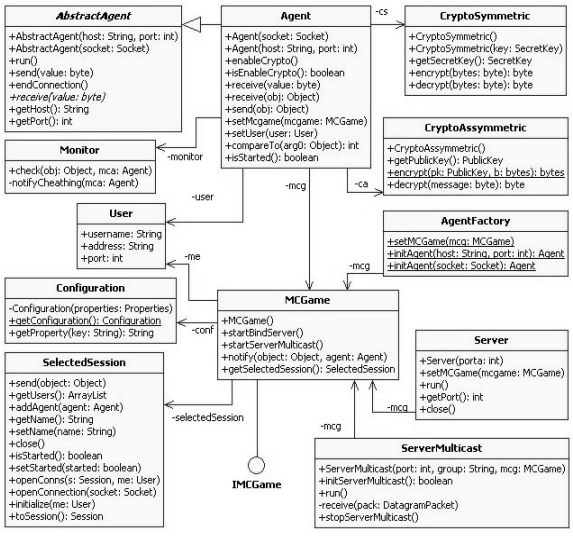


Figura 1: Diagrama de classes do middleware.

3.1 Interfaces de integração

O middleware MCommP2P foi idealizado para ser integrado facilmente a qualquer jogo em rede que esteja preparado para isso. Essa integração pode ser feita diretamente, com chamadas explícitas vindas do código funcional do jogo. Para isto, duas interfaces foram especificadas no middleware: Observer, que deve ser implementada pelo jogo, e IMCGame, que corresponde aos serviços de comunicação do MCommP2P. A Figura 2 apresenta um digrama de componentes dessa integração.

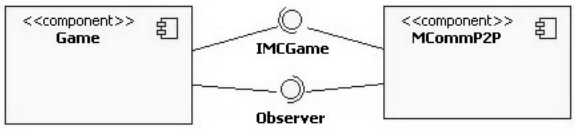


Figura 2: Modelo de integração do middleware.

Outra opção de integração é através de uma incorporação a um dos motores de jogos disponíveis. De fato, o middleware foi idealizado para utilização em jogos com o padrão arquitetural semelhante ao do motor de jogos JFROG [Bittencout et al. 2003], cujo modelo dá ênfase a modularização do jogo. Nestes tipos de jogos a parte de comunicação é separada do restante do jogo, fazendo com que o middleware possa ser integrado com este através dos serviços disponibilizados e de uma interface para notificação de eventos ao módulo de controle do jogo. Apesar disso, o MCommP2P pode ser adicionado a qualquer arquitetura de jogo, desde que o programador trate eventuais necessidades de compatibilização. A Figura 3 apresenta uma idéia de como o middleware MCommP2P poderia ser integrado ao modelo arquitetural do motor de jogos JFROG.

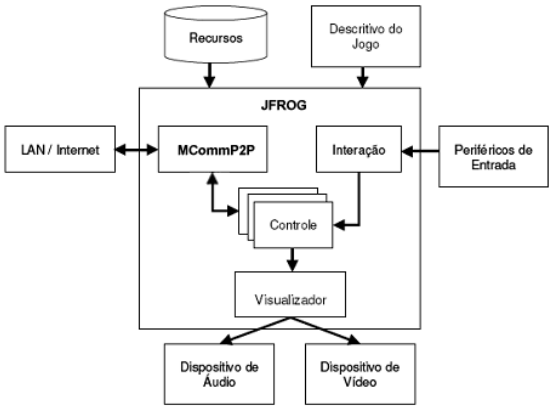


Figura 3: JFROG com o middleware MCommP2P.

3.2 Casos de testes

Para verificar a usabilidade do middleware no desenvolvimento de jogos em rede, foram desenvolvidos dois protótipos de jogos que ilustram as possibilidades da aplicação: Jogo da Velha e Space Invaders (jogo clássico na vertical de naves espaciais, onde o objetivo do jogo em rede é uma nave acertar a

outra). Para o desenvolvimento do Jogo da Velha, foi criada uma única classe contendo tanto a lógica do jogo quanto sua interface. Este jogo teve como objetivo principal realizar os primeiros testes e ajustes do middleware, não sendo analisadas características relacionadas à performance. Ao final dos ajustes, diversas funcionalidades já estavam validadas, como a inicialização do jogo (gerência de usuários e sessões) e a troca de mensagens.

O desenvolvimento do jogo Space Invaders necessitou de mais planejamento, visto que sua finalidade foi testar não somente o funcionamento, mas também o desempenho do middleware desenvolvido, além de requisitos mais complexos como a sincronização de estados do jogo. O modelo estrutural do motor JFROG foi utilizado como base para a definição arquitetural do jogo, como apresenta a Figura 4. Neste jogo foram definidas três classes principais, sendo elas: CaptureKeyboard, responsável pela captura de entradas do teclado, Canvas, responsável pela renderização gráfica do jogo, e Game, que realiza a consistência e dinâmica de estados do jogo.

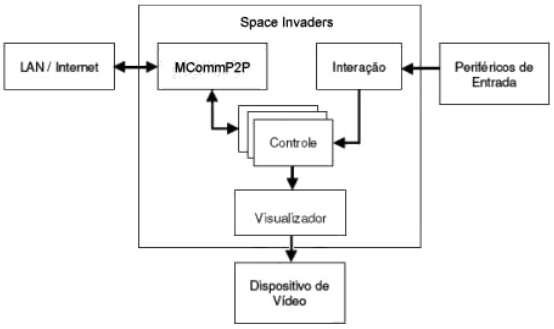


Figura 4: Modelo conceitual do jogo, com o MCommP2P.

Para realização dos testes iniciais de desempenho, o middleware foi configurado com as opções de sincronização de estados e criptografia ativas, e o jogo teve sua taxa de atualização fixada em 30 vezes por segundo. Esta taxa indica que o estado do jogo é atualizado 30 vezes a cada segundo, implicando na sua alteração a partir da captura de eventos do teclado e mensagens da rede, e renderização para apresentação ao jogador. Com essa configuração apresentada o jogo manteve-se estável durante os testes, sem ocorrência de erros, a uma taxa de atualização de quadros em 30 frames por segundo (fps) e um consumo de banda médio de 24 Kbps.

Para os jogos utilizados nos testes, a união das técnicas de sincronização, serialização e criptografia não geraram atrasos visivelmente perceptíveis em ambos os casos. Em relação ao tamanho dos pacotes, a técnica de criptografia gerou um aumento médio de 10%. Já na serialização, o aumento médio foi de 15% mais 40 bytes, referente ao cabeçalho de identificação da classe. Para pacotes muito pequenos é aconselhado o uso de arrays de bytes, pois neste caso a serialização gera somente um aumento de 25 bytes referente ao cabeçalho de identificação.

4. Considerações finais

O trabalho desenvolvido apresentou um middleware P2P para suporte ao desenvolvimento de jogos em rede. Esse middleware possui grande potencial para suporte ao desenvolvimento de jogos em rede, sobretudo quando consideramos pequenos desenvolvedores. Os testes iniciais mostraram que o MCommP2P funciona como esperado, podendo ser utilizado em projetos pilotos.

Como trabalhos futuros, pretende-se implementar o middleware sobre plataformas avançadas, como o XPastry e JXTA (MCommP2P foi montado sobre sockets TCP). Além disso, é esperada a integração do middleware com jogos mais complexos, visando a realização de testes mais conclusivos sobre a usabilidade comercial da solução. Por fim, testes formais de desempenho serão realizados, buscando a comparação do MCommP2P com outras soluções.

Referências

Bittencourt, J. R. e Osório, F. S. 2006. “Motores para Criação de Jogos Digitais: Gráficos, Áudio, Interação, Rede Inteligência Artificial e Física”. Disponível em: http://osorio.wait4.org/oldsite/iajogos/artigos/bittencourt-osorio-eri-mg2006.pdf. Porto Alegre.

Azambuja, J. 2006. “Peer to Peer: Modelos, Middlewares e Aplicações”. Disponível em: http://www.inf.ufrgs.br/~asc/sodr/pdf2006-02/SODRJoseAzambuja.pdf. Porto Alegre.

Jábali, S. 2004. “Introdução ao Projeto Peers”. Disponível em: http://www.lsd.ic.unicamp.br/projetos/peers/arquivos/PEERS_Intro.pdf. Laboratório de Sistemas Distribuídos, Unicamp. Campinas.

Anibolete, T. J. 2006. “Algoritmos distribuídos em Jogos Multi-usuário”. Disponível em: http://www-di.inf.puc-rio.br/~endler/courses/DA/Monografias/06/MultiUserGames-Tulio-Mono.pdf. PUC-RIO. Rio de Janeiro.

Kozovits, L. E. 2003. “Arquiteturas para Jogos Massive Multiplayer”. Disponível em: ftp://ftp.inf.puc-rio.br/pub/docs/techreports/03_36_kozovits.pdf. Rio de Janeiro.

Hinz, M. A. M. 2000. “Um estudo descritivo de novos algoritmos de criptografia”. Disponível em: http://www.ufpel.tche.br/prg/sisbi/bibct/acervo/info/2000/Mono-MarcoAntonio.pdf. Pelotas.

Baughman, N. E, e Levine, B. N. 2001. *Cheat-proof playout for centralized and distributed online game*. In: Procedures of IEEE Infocom, San Francisco.

Bittencourt, J. R., Giraffa, L. M. M. e Santos, R. C. 2003. *Criando Jogos Computadorizados Multiplataforma com Amphibian*. In: II Congresso Internacional de Tecnologia e Inovação em Jogos Computadorizados. Curitiba.

Métricas para Avaliação de Motores de Jogos Tridimensionais

Ítalo Mendes da S. Ribeiro
italo.ribeiro@gmail.com

Selan Rodrigues dos Santos
selan@dimap.ufrn.br

Departamento de Informática e Matemática Aplicada
Universidade Federal do Rio Grande do Norte
Campus Lagoa Nova, 59072-970, Natal/RN

Resumo

Existe uma variedade de motores de jogos tridimensionais (3D) disponíveis para diversos objetivos, como a criação de jogos, simulações, animações entre outros. Contudo, também existe uma grande necessidade de definição de mecanismos que permitam realizar comparações entre os motores, levando em consideração, por exemplo, parâmetros como qualidade de software, funcionalidades suportadas, facilidade de uso. Outro fator determinante que dever ser levado em consideração em uma eventual comparação de motores é a natureza da aplicação que se deseja desenvolver com o auxílio do motor. Neste artigo propomos um sistema de métricas para a avaliação e comparação de motores de jogos. A título de validação, aplicamos este sistema de métricas na avaliação de dois populares motores de jogo 3D.

Keywords: motores de jogo 3D, métricas, desenvolvimento de aplicações gráficas.

1 Introdução

Os primeiros jogos eletrônicos eram bem simples e possuíam uma programação monolítica e dependente da plataforma. O lançamento dos títulos Doom e Quake [id software 2009] foi um momento marcante na evolução dos jogos eletrônicos, por apresentar um maior realismo gráfico e sonoro, com ações mais realistas, se comparado aos títulos da época. Isso foi possível devido ao aumento de desempenho dos computadores em geral, e o aprimoramento de hardware especializado, como é o caso das placas de aceleração gráfica com processadores dedicados (GPUs).

Outra grande contribuição destes títulos foi a introdução do conceito de *motores de jogos 3D*. Os motores podem ser considerados como uma camada extra de abstração separando a camada de aplicação dos comandos em mais baixo nível e específicos, como renderização gráfica, inteligência artificial e conexão em rede. A principal vantagem desta nova abordagem é a possibilidade de reutilizar o mesmo motor para títulos similares, bastando modificar a camada da aplicação.

Essa evolução conjunta de software (com os motores) e de hardware (com o hardware dedicado de baixo custo) foi responsável pelo notável crescimento da indústria de desenvolvimento de jogos. Atualmente os motores são empregados principalmente na criação de jogos eletrônicos, simulações físicas, médicas, composição de alguns elementos de animações, passeios virtuais interativos, etc. [Lewis and Jacobson 2002]

Contudo, essa mesma evolução trouxe consigo alguns novos desafios, como por exemplo um progressivo aumento na complexidade de desenvolvimento de jogos. Isso se deve, por exemplo pela a adição de mais elementos aos jogos, como simulação de Física, comunicação de grande quantidade de dados entre milhares de jogadores em rede, construções de ambientes 3D massivos, segurança na transmissão dos dados, etc. [Rollings and Morris 2003] Para superar estes desafios, foram desenvolvidos um grande número de motores de jogos com funcionalidades diversas, o grande obstáculo, portanto, passa a ser definir que motor é adequado para uma determinada aplicação.

Muitas vezes a escolha de um motor é feita com base no seu custo ou puramente na experiência pessoal prévia dos desenvolvedores, ou seja, sem critérios objetivos e bem definidos. Desta forma, a proposta deste trabalho é apresentar um sistema de métricas para auxiliar na tomada de decisão com relação a escolha do motor mais apropriado para uma determinada aplicação. Para tanto, o sistema de métricas leva em consideração as diversas funcionalidade dos motores, ponderado pela categoria de aplicação a ser desenvolvida.

2 Trabalhos relacionados

A utilização de métricas não é novidade. Por exemplo, em [Eves and Meehan 2008] são apresentadas algumas métricas com pontuações associadas. No entanto, a pontuação é definida por um analisador que está aplicando as métricas para um determinado projeto para a construção de uma aplicação 3D. Isso pode tornar o resultado imparcial e variável, pois outra pessoa pode aplicá-las no mesmo projeto e os resultados serem muito diferentes.

O trabalho apresentado em [Korva 2007] cita muitos pontos que devem ser considerados para a escolha de um motor de jogos 3D, mas não define uma pontuação. Já a métrica proposta por Threnholme e Smith [Trenholme and Smith 2008] possui poucos pontos a serem analisados, não abrangendo de forma satisfatória todas as funcionalidades que um motor de jogos possui e a comparação entre os motores é realizada principalmente a partir do uso do motor, tornando a comparação pouco precisa.

Assim, a criação de um modelo de comparação entre motores de jogos 3D que aborde os vários aspectos (gráficos, acessórios, suporte e de software) existentes nessas ferramentas é necessário para que uma análise completa seja feita. A utilização de uma pontuação para cada métrica é útil para uma medida mais precisa de cada motor de jogos. Além disso, a pontuação pré-definida considerando a facilidade que o quesito adiciona a criação de aplicações 3D, evita que o resultado seja influenciado pela opinião de cada avaliador, eliminando as diferenças entre os resultados obtidos.

3 Métricas Propostas

Alguns elementos devem ser levados em consideração para a comparação entre motores de jogos. As suas funcionalidades são os principais fatores a serem avaliados, podendo ser classificadas em cinco categorias: *gráficas*, *acessórias*, *suporte*, *software* e *aplicações*. As funcionalidades gráficas compreendem os recursos que o motor dispõe para a renderização do ambiente. Nesta categoria se enquadram elementos como a geração de terreno, *skybox*, efeitos especiais como neblina, fogo e *flare*, texturas suportadas, geração de sombras, iluminação, etc.

Dentre as funcionalidades acessórias podem ser citadas a simulação de Física, muito importante para aumentar o realismo e dinamismo do ambiente; a conexão em rede, que possibilita a realização de partidas online através da rede de computadores; e a inteligência artificial, responsável por controlar os NPCs (*non-player character*), por exemplo. As funcionalidade relacionadas à suporte correspondem à documentação, suporte e a atividade da comunidade de desenvolvedores e utilizadores, uma vez que isso pode contribuir para a solução de problemas com o uso do motor.

As linguagens suportadas e as plataformas de execução de um motor são outros fatores importantes de comparação. Um outro elemento a ser considerado é a estruturação do motor, que acaba determinando tanto a forma de construção dos projetos quanto a facilidade de aprendizagem dele. Essas funcionalidades compõem as métricas de software.

Do conjunto de métricas citadas, algumas são mais importantes que outras em alguns tipos de aplicações. Por exemplo, uma aplicação de simulação possui necessidades bem distintas das de um jogo com finalidade de entretenimento. Dessa maneira, algumas métricas devem ser ressaltadas em relação a outras, dependendo da aplicação em desenvolvimento.

As métricas de qualidade de software como portabilidade, eficiência e usabilidade estão relacionadas às métricas que serão apresentadas direta ou indiretamente, através das plataformas suportadas pelos motores (portabilidade), da documentação (usabilidade) e facilidade de criação dos elementos do ambiente (eficiência).

Cada critério da métrica recebe uma pontuação variando de $[0; 3]$, de acordo com o grau de dificuldades tecnológica para a existência do recurso no motor e a facilidade que ela introduz na criação de uma aplicação 3D. A categorização da pontuação é descrita a seguir:

0 - Inexistente: recurso não presente no motor.
1 - Básica: recurso essencial, que deve estar presente em praticamente todo motor.
2 - Recomendada: recurso intermediário mas que aumenta a produtividade da aplicação e conduz a um melhor efeito final para aplicação; presente na maioria dos motores.
3 - Avançada: recurso existente em poucos motores, normalmente associado a efeitos e características específicas, que diferenciam o motor dos demais; muito útil no desenvolvimento.

Por exemplo, a presença de algum tipo de gerente de cena recebe pontuação 1, enquanto que se o mesmo motor apresentar uma técnica de otimização de cenário, como *octree* ou paginação de terreno, receberia nota 2.

O cálculo total da pontuação de um motor é o somatório de todos os pontos obtidos nas diversas categorias da métricas. As métricas que são incorporadas em um motor na forma de complemento (*add-ons*) também entram na contagem dos pontos. A Tabela 1 contém todas as métricas e suas respectivas pontuações, localizada ao lado de cada métrica.

As métricas onde a pontuação difere da forma estabelecida no início desta seção são marcadas com um '*' na Tabela 1 e serão discutidas a seguir.

3.1 Formatos Suportados

A pontuação deste quesito é indicado de acordo com a quantidade de formatos de imagens, malhas e áudio suportados pela motor: **(1)** - 1 à 5 formatos; **(2)** - 6 à 10 formatos; e **(3)** - acima de 10 formatos.

3.2 API Gráfica

Um motor de jogos é construído sobre uma API gráfica, que é um conjunto de funções para acessar os componentes gráficos do dispositivo. As duas principais APIs gráficas utilizadas são OpenGL e DirectX. A pontuação é de acordo com as APIs suportadas pelo motor: **(1)** - Somente OpenGL ou DirectX; e **(2)** - OpenGL e DirectX.

3.3 Status

De acordo com o estado de desenvolvimento a pontuação adicionada para a avaliação é a seguinte: **(1) - Inativa:** o motor não está mais sendo mantido ou atualizado; **(2) - Beta:** o motor ainda está em um estágio de testes; e **(3) - Estável:** o motor atingiu um estado estável, e pode ser utilizada para desenvolvimento sem problemas.

3.4 Custo

O custo, em muitos casos, é um fator decisivo na escolha do motor. Existem muitos motores que apesar de possuírem ótimas ferramentas e muitas funcionalidades, têm custos elevados, tornando-os inacessíveis para um único usuário ou pequenas empresas de desenvolvimento.

Dessa maneira, motores gratuitos ou de baixo custos recebem as maiores pontuações. A pontuação atribuída ao motor nesta categoria depende da faixa de valores em que a sua licença se enquadra: **(1)** - acima de US$10000; **(2)** - entre US$3000 e US$9999; **(3)** - entre US$1000 e US$2999; **(4)** - entre US$1 e US$999; e **(5)** - gratuita.

Para um mesmo motor a licença pode variar de gratuitas com restrição de uso para algumas funcionalidades, até o preço completo, onde todas as suas funcionalidades estão disponíveis. Portanto, o processo de pontuação deve considerar apenas as funcionalidades presentes no motor de acordo com a licença escolhida.

3.5 Sistema Operacional

A quantidade e os tipos de sistemas operacionais (SOs) para os quais um motor de jogos pode gerar uma aplicação são importantes para os planos de negócio de uma empresa. O suporte do motor a diversas plataformas permite a definição dos dispositivos para os quais a aplicação poderá ser distribuída. Este fato influencia diretamente a abrangência da quantidade de usuários e consumidores da aplicação.

Os tipos de SOs suportados foram organizados de acordo com os dispositivos suportados: *computadores* (Windows, Linux(Unix), MacOS); *consoles* (GameCube, Nintendo Wii, Nintendo DS, modelos Playstation, modelos Xbox); e *dispositivos móveis* (iPhone, smartphones, pocket pcs).

A pontuação do tipo de SO é atribuída da seguinte forma: **(1)** - suporta somente um dos tipos de sistemas operacionais; **(2)** - suporta dois tipos; e **(3)** - suporta os três tipos.

A pontuação relacionada a métrica de quantidade de SO é adicionada conforme o número de SOs suportados pelo motor.

3.6 Linguagem de Programação

A linguagem de programação é um fator importante para a escolha do motor de jogos. Dependendo do tipo de linguagem utilizada, pode-se gerar aplicações com desempenho variável.

Quanto mais linguagens um motor de jogos suportar, maiores são as possibilidades de aplicações que podem ser geradas. Assim a pontuação desse quesito reflete a quantidade de linguagens suportadas.

4 Análise por tipo de aplicação

De acordo com o tipo de aplicação que será desenvolvida com o motor de jogos, algumas métricas tem uma maior importância em relação as outras, na facilitação da criação da aplicação. Isso deve influenciar na escolha do motor. Cada métrica apontada nos tipos de aplicações deve ter 3 pontos adicionados aos já estabelecidos anteriormente. Por exemplo, se 1 ponto é normalmente atribuído à métrica *shadow mapping* e este critério é relevante para um certo tipo de aplica, a pontuação deve ser acrescida de mais 3 pontos, totalizando 4 para o quesito.

Tabela 1: *Métricas e suas respectivas pontuações, onde as letras ao lado do nome do grupo de métricas são G(gráficas), A(acessórios), Sp(suporte) e Sf(software).*

Gerente de cena (G)		Iluminação (G)		Efeitos Especiais (G)		Texturização (G)		Animação (G)		Malhas (G)	
Geral	1	*Lightmaps*	1	*Billboarding*	1	Básica	1	*Keyframes*	2	Leitura	1
BSP	1	Por-Vértice	1	*Lens flare*	2	Multitextura	1	Por esqueletos	2	Deformação	3
Portais	2	Por-Pixel	2	Sist. partículas	2	*Bump mapping*	2	*Blending*	2	Progressivas	3
Octrees	2	Anisotropia	2	Prof. de campo	2	Formatos	*	*Inv. knematics*	3	Formatos	*
Pag. paisagem	3	Volumétrica	3	Névoa ou Céu	2			*Fwd. knematics*	3		
		Radiosidade	3	Água ou Fogo	3			*Morphing*	3		
		BRDF	3	*Motion blur*	3			Facial	3		
				Climáticos	3						

Shaders (G)		Terreno (G)		Sombras (G)		API Gráfica (G)		Sist. Op. (Sf)		Linguagens (Sf)	
Vertex shader	2	Básico	1	*Shadow map*	1	OpenGL	*	Tipo	*	C/C++,	
Pixel shader	2	Deformável	3	*Shadow volume*	3	DirectX	*	Quantidade	*	Java, etc.	*

Gerais (G)		Rede (A)		Física (A)		Int. Artificial (A)		Som (A)		Suporte (Sp)	
GUI	1	Estruturação	1	Básico	1	*Pathfinding*	2	Básico	1	Exemplos	1
Entrada/saída	2	Cliente/servidor	2	Detecção colisão	1	Tomada decisão	2	3D	2	Fórum	2
Scripting	2	Ponto a ponto	2	Veículo	3	*Script*	2	*Streaming*	2	*Chat online*	2
Possui algum		Segurança	3	*Ragdoll*	3	Formação	3	Formatos	*	Documentação	2
tipo de editor	3			Terreno	3						
Estado desenv.	*			Fluído	3						
Custo	*										

4.1 Educacionais

As aplicações educacionais objetivam auxiliar no processo de aprendizado dos seus usuários sobre um determinado assunto [Bellotti et al. 2009]. Normalmente este tipo de aplicação não necessita de muitos recursos computacionais para sua execução, pois pretende-se atingir o maior número de possível de usuários. A iluminação, sombras, terrenos e muitos outros efeitos especiais, adicionados por *shaders*, normalmente não são essenciais em projetos desse tipo. Além disso, não costumam utilizar recursos de comunicação em rede, nem componentes de física ou de inteligência artificial, embora isso não signifique que eles não sejam empregados em jogos educacionais específicos.

Assim, as métricas que devem ser acentuadas para aplicações educacionais são: *(1)* **Sombras:** *shadow volume*; *(2)* **Texturização:** multitextura; *(3)* **Animação:** *keyframes*, baseada em esqueletos, *blending*, facial; *(4)* **Efeitos especiais:** *billboarding*, sistemas de partículas; *(5)* **Gerais:** *scripting*, editor de partículas, interface gráfica com usuários (GUI); e *(6)* **Som:** som 3D.

4.2 Simuladores

Os simuladores procuram criar, o mais próximo possível da realidade, determinadas atividades em um ambiente 3D [Stang 2003]. São muito utilizados para estudos científicos realizando a análise dos resultados em várias situações que poderiam acontecer no mundo real, como por exemplo no treinamento de profissionais através dos simuladores de cirurgias [Marks et al. 2007]. Os jogos desse gênero costumam fazer grande sucesso no público adulto como o simulador de vôo *Flight Simulator* [Microsoft 2009].

São aplicações que contém muitos detalhes em sua composição, utilizam muitos recursos computacionais e possuem ótima qualidade gráfica. Muitas das funcionalidades de um motor são importantes no seu desenvolvimento. *(1)* **Gerenciamento de cena:** particionamento binário do espaço (BSP), *octrees*, portais, paginação de paisagem; *(2)* **Iluminação:** anisotropia, volumétrica, radiosidade, BRD; *(3)* **Sombras:** voluma de sombras; *(4)* **Texturização:** multitextura, *bump mapping*; *(5)* **Shaders:** *vertex* e *pixel shaders*; *(6)* **Malhas:** deformação; *(7)* **Efeitos especiais:** *lens flare*, sistema de partículas, profundidade de campo, céu, água, fogo, névoa; *(8)* **Terreno:** deformável; *(9)* **Gerais:** GUI; editores de *script*, caminhos, terreno, partícula; *(10)* **Som:** som 3D e *streaming*; *(11)* **Física:** detecção de colisão, veículo, *ragdoll*, fluídos, corpos rígidos; e *(12)* **Inteligência artificial:** *pathfinding*

4.3 Visualização

Esse tipo de aplicação serve para a visualização de ambientes 3D, ou reconstruções tridimensionais a partir de imagens médicas bidimensionais. Podem também ser utilizadas para a visualização e manipulação de informações ou dados com muitas dimensões [Harrop and Armitage 2006].

Os visualizadores são aplicações em que o principal foco esta na forma como será feita a visualização dos ambientes. Portanto, recursos como shaders, efeitos especiais, e inteligência artificial não são imprescindíveis para aplicações desse tipo. Assim métricas importantes são: *(1)* **Sombras:** voluma de sombras; *(2)* **Texturização:** multitextura, *bump mapping*; *(3)* **Animação:** *keyframes*, baseada em esqueletos; *(4)* **Efeitos especiais:** *lens flare*, sistema de partículas, profundidade de campo, céu, água, fogo, névoa; *(5)* **Gerais:** interface gráfica com usuário e suporte a vários formatos de imagem; *(6)* **Som:** som 3D e *streaming*; *(7)* **Física:** detecção de colisão, veículo, *ragdoll*, fluídos, corpos rígidos; e *(8)* **Rede:** conexão cliente-servidor ou *multi-cast*.

5 Avaliações

Afim de demonstrar a utilização das métricas, elas foram utilizadas para a avaliação de dois motores de jogos populares.

5.1 OGRE

A OGRE (*Object-Oriented Graphics Rendering Engine*) é um motor voltado para a manipulação gráfica escrito em C++, não sendo um motor de jogos completo [Junker 2006]. Quando é necessária a utilização de outros recursos como áudio e física por exemplo, outras bibliotecas podem ser unidas ao OGRE através de seus *add-ons* que facilitam a integração. É um motor gratuito, de código aberto, multiplataforma e em pleno desenvolvimento. Possui versões em outras linguagens como Java e Phyton. A comunidade de usuários é grande e muito ativa.

A OGRE possui ótimos gráficos e muitos efeitos visuais, mas não possui bons editores. Os elementos na cena são organi-

zados em uma estrutura hierárquica de nós de cena. A documentação é bem completa e possui uma grande contribuição da comunidade, embora haja a necessidade de atualizações nos textos disponíveis.

A OGRE conseguiu 116 pontos dos 177 que poderia obter caso obtivesse pontuação máxima em todas as métricas, conseguindo assim 65,53% da pontuação total. Nos tipos de aplicações conseguiu 152 (116 + 32) pontos nas educacionais, onde obteve 32 dos 42 pontos que poderiam ser obtidos (85,71%). Nos simuladores 209 (116 + 93), com 70,45% dos 132 possíveis e 146 (116 + 30) nas de visualização, onde obteve 90,90% do máximo que é de 33 pontos.

5.2 Irrlicht

A Irrlicht é um motor escrito em C++ simples e rápido, muito bom para os iniciantes na área de desenvolvimento de jogos, pois com poucas linhas de código é possível construir uma aplicação [Gebhardt 2009]. Possui uma documentação básica, embora seja um bom começo para os novo usuários. É também gratuita e de código aberto.

A organização dos componentes da cena é semelhante a OGRE, utilizado nós de cena. A comunidade é bem ativa com muitas participações no fórum e no desenvolvimento de complementos para o motor. Possui um editor de ambiente 3D nativo, o `IrrEdit`, qual possui muitos dos elementos necessários para a construção da cena que podem ser trabalhados, como luzes, objetos e terreno. O motor possui um sistema de detecção de colisão simples, um editor de interface com usuário. As bibliotecas para facilitar a integração são a `IrrNet` (usa Enet) para conexão em rede, a `IrrAI` para inteligência artificial `IRRklang` para áudio.

A avaliação da Irrlicht resultou na pontuação 104 representando 58,75% do total de pontos que poderia obter. Para aplicações educacionais conseguiu 137 (106 + 33) pontos, aproveitamento de 75%, simulações 179 (106 + 75), com 56% do total e nas de visualização 131 (106 + 27), obtendo 81,81%.

5.3 Discussão dos resultados

Observando os resultados da OGRE e da Irrlicht, a primeira mostrou-se um melhor motor com uma pontuação um pouco superior a da segunda. Um dos principais diferencias que aumentaram os pontos da OGRE em relação a Irrlicht foi a quantidade de efeitos visuais oferecidos por ela. Isso se deve devido a OGRE ser, em sua essência, um motor gráfico. Um outro fator determinante foram funcionalidades encontradas somente na OGRE, como a animação facial.

A Irrlicht é um motor mais voltado para o desenvolvimento de jogos, mais simples e prático de ser utilizado. Oferece um conjunto de editores que agilizam o processo de desenvolvimento e que juntamente com o suporte a inteligência artificial ajudaram no aumento da sua pontuação e são um diferencial em relação a OGRE.

Nas aplicações educacionais e de visualização os dois motores mostraram-se ótimas opções, com pontuações muito próximas e boas (152 e 137, 146 e 131). Para aplicações de simulação a OGRE mostrou-se melhor devido aos efeitos visuais oferecidos, pois aumenta o realismo visual de uma simulação.

6 Conclusão e trabalhos futuros

A avaliação de motores ainda é um processo em desenvolvimento. Assim, as métricas propostas procuram criar uma referência inicial para a comparação entre os motores de jogos 3D, que são utilizados para os mais diversos tipos de aplicações 3D. De acordo com a aplicação a ser desenvolvida, algumas métricas tem uma maior importância em relação a outras, e mesmo assim ajustes são necessário para casos específicos.

O processo de avaliação procura analisar os motores de forma precisa e padronizada, sem muitas variações como ocorre na maioria das comparações realizadas em trabalhos anteriores, nos quais normalmente a análise é baseada na experiência do uso do motor pelo autor. Como proposto, não importa quem realize a avaliação a pontuação da OGRE e da Irrlicht serão as mesmas ou muito próximas (claro, assumindo que os pesos dos critérios são os mesmos).

Com a natural evolução das motores de jogos novas métricas podem ser adicionadas, e outras terem sua pontuação modificada. Novas plataformas e tipos de aplicações também devem serem consideradas e incluídas nas métricas, acomodando critérios específicos. Por exemplo, os jogos para celulares possuem características particulares, onde fatores como consumo de energia, uso restrito de memória e poder de renderização limitados devem ser considerados.

O próximo passo será o aperfeiçoamento das métricas propostas, aplicando-as a diversos motores, para então validar os resultados obtidos.

Referências

AUTODESK, 2009. 3d studio max. `http://www.autodesk.com/3dsmax`, acessado em maio 2009.

BELLOTTI, F., BERTA, R., GLORIA, A. D., AND PRIMAVERA, L. 2009. Enhancing the educational value of video games. *Comput. Entertain. 7*, 2, 1–18.

EVES, G., AND MEEHAN, P., 2008. The development of selection criteria for game engines in the development of simulation training systems.

GEBHARDT, N., 2009. Irrlicht engine. `http://irrlicht.sourceforge.net/`, acessado em junho 2009.

HARROP, W., AND ARMITAGE, G. 2006. Modifying first person shooter games to perform real time network monitoring and control tasks. In *NetGames '06: Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games*, ACM, 10.

ID SOFTWARE, 2009. Quake. `http://www.idsoftware.com/games/quake/quake/`, acessado em maio 2009.

JUNKER, G. 2006. *Pro OGRE 3D Programming*. Apress.

KORVA, T., 2007. Open source & low cost game engines.

LEWIS, M., AND JACOBSON, J. 2002. Game engines in scientific research - introduction. *Communications of the ACM 45*, 1, 27–31.

MARKS, S., WINDSOR, J., AND WÜNSCHE, B. 2007. Evaluation of game engines for simulated surgical training. In *GRAPHITE '07: Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia*, ACM, New York, NY, USA, 273–280.

MICROSOFT, 2009. Microsoft flight simulator. `http://www.fsinsider.com/`, acessado em maio de 2009.

ROLLINGS, A., AND MORRIS, D. 2003. *Game Architecture and Design*. New Riders, Pearson Education.

STANG, B. 2003. Game engines: Features and possibilities. Tech. rep., Institute of Informatics and Mathematical Modeling at The Technical University of Denmark.

TRENHOLME, D., AND SMITH, S. P., 2008. Computer game engines for developing first-person virtual environments.

Multiagent Gathering in Real-Time Strategy Games

José Carlos Moura Júnior Geber L. Ramalho

Centro de Informática – Universidade Federal de Pernambuco

Abstract

One common task in Real-Time Strategy Games (RTS) is resource gathering (for instance, gold mining or gas extraction). This task is usually performed by some Non Player Characters (NPCs), which are implemented as intelligent agents. In order to maximize resource gathering by a team of agents, coordination is required. In this paper, we present an experimental work concerning the study of various architectures of multi-agent systems devoted to resource gathering. The pros and cons of each architecture is discussed according to some evaluation criteria we have proposed.

Keywords: resource gathering, foraging, multi-agent, RTS games.

Authors' contact:
{jcmj,glr}@cin.ufpe.br

1. Introduction

RTS games are a very popular game genre, and have an interesting domain to be studied by the Artificial Intelligence area, due to the challenges that the game has to offer. Many AI techniques can be used to solve the problems as combat, team coordination and patrolling. An interesting challenge for IA found on RTS Games is the resource gathering, a multi-agent task that needs to be coordinated.

The problem that will be analyzed here is part of a competition that happens since 2006 in AIIDE 2006 (Artificial Intelligence and Interactive Digital Entertainment). In this competition the participants should develop a group of agents that collects a bunch of resources, spread in the environment, and take them back to the Control Center, like the gathering that happens in a RTS game.

Multi Agent gathering is an area with many studies and techniques, but in RTS context still is an open subject. Despite the current studies on multi-agent gathering, none of them can be seen as a definite solution to the problem. In other words, multi-agent gathering is a problem that needs to be more explored and studied.

This paper presents a performance evaluation, and analysis, of several multi-agent architectures to this important task for RTS games.

2. Gathering

Gathering, or foraging, is the act of searching for food and provisions, and collect then in an area. The foraging have some characteristics that helps to define it:

- Mono Agent X Multi Agent: In mono agent foraging a single agent is responsible for collect the resources, while in the multi-agent a group of agents is responsible to do the gathering.
- Central Place X Various Place: It refers to the amount of spots where the collected objects should be deposited. In central place, there's only one sink, where all the collected objects should be put. In the second case there's two or more places where the collected objects should be deposited.
- Recurrent X One Shot: One Shot means that the agent needs to visit each resource only one time, in recurrent foraging, the agent needs to visit the same resource several times to collect the entire resource.

2.1 Gathering in RTS

In a RTS game the first problem the player will have is the foraging or resource gathering. The player needs to collect resources so he can create buildings, evolve new technologies, train units, etc. Teams with a more efficient resource gathering will have a greater amount of resources, so they will have more advantage in the battle.

Resource gathering is the basis of the economy in a RTS society, so some combat tactics are based on foraging to collect the most of resources as fast as possible to evolve; or attacks the enemy foraging points that will make his economy weaker.

Raiding for instance is a combat tactic that his main objective is to make constant attacks to the foraging enemy points, because the forager units have low combat power. It usually doesn't wins the game, but make it more fragile.

The Fast-Heroic tactic, is based on the fast technologic evolution, because if a society is more evolved, so your units will be. To achieve the resources needed to the evolution, it doesn't create combat units, it only creates forage units so they can collect the resources as fast as it's possible, allowing that way the evolution.

Foraging is a complex problem that needs to use the maximum capacity of each agent, making then to carry as many resources as they can, so they can collect it as soon as possible.

The RTS Games foraging, and that will be analyzed, can be described as Multi Agent Central Place Recurrent Foraging, and this environment is fully observable, deterministic, sequential, dynamic and

continuous. Multi Agent because a group of agents will collect the resources, and there's a need to create a coordination among then. Central Place, because there's a Control Center and all the resources gathered should be taken there. Recurrent, the agent needs to visit each mine several times to collect the resource. The environment is fully observable, each agent have complete information about the environment. The world next state is complete consequence of the actions in the present moment so the environment is deterministic.

The resource gathering can be disassembled in 2 main problems, "Where should I go?" (Resource allocation) and "How do I get there?" (Path-Finding), and later in various others problems studied by the Artificial Intelligence.
The resource allocation is about, what resource each agent should be responsible for collect. And a series of points should be taken in consideration, for instance:

- Distance to the resource: How far is the resource to the Control Center, the most interesting are those that are closer to the Control Center, so the agent could collect, go back to control center and then go to the resource again in a few time.
- The resource availability: It's necessary that the resource is available, there should de a way to get to the resource and no other agent should be collecting the resource.

Path-finding is the most popular, potentially frustrating, problem that can be found in the Artificial Intelligence in games. It's desirable that the path found be as less expensive as possible, in the time to search it and cost to execute it.

On point that can interfere in the resource allocation decision and in the path-finding decision is the coordination among the agents that will collect the resources; it needs some synchronization, so they won't get in the way of each other. It needs to keep in mind that each decision should be taken with the objective to get the best result to the team and not the best for each agent individually.

Another multi-agent gathering problem is the communication; it should be done in an efficient way so each agent can inform the others from his team his decisions, because his decision can interfere in the other agent decision. For instance, if an agent decides to go to a mine the others agents should be informed so any other agent would go there, avoiding this way a unnecessary travel to the mine.

3. State of the art

In this section it will be analyzed the different studies in multi-agent gathering, and how they fit or not in the specific RTS gathering:
Many strategies where described in different articles about foraging, most of them doesn't fit to the exactly problem, from AIIDE 2006, [OSTEGAARD], [SHELL], [SIMONIN], [HAYAT], [WANG],[STEELS] and [SUGAWARA].

4. Methodology

The objective of this work, is to analyzing different approaches to the problems mentioned in the previous section.

The first parameter that should be considered is the *basic type of the agent*: reactive or cognitive. Reactive agents take his decisions based only in its current world perception; it cannot execute plans (scheduled actions). Cognitive agents can consider in its decision making events that did not happen yet, and can create and execute plans.

The second parameter is the *Path-Finding technique.* In this work two options will be analyzed: the first one is the A* (A-Star) Algorithm, the most used path-finding algorithm, because it guarantees to find the best path between two points if it exists [BOURG]. The other path-finding option is the LRA* (Local Repair A*), a modification of A* for multi-agent environments. [SILVER]. In LRA*, the agent creates its path ignoring all the other agents, except his current neighbors, the agent moves along its path until find some other agent, when it founds another agent blocking its path, it calculates his path from current point.

The next point to consider is the *coordination strategy*. Firstly we will use a central coordinator, a sort of oracle, which controls the gathering agents. The other coordination strategy we will study is decentralized, where the main decisions concerning where and when to gather and when are taken by each agent.

Another parameter concerns communication. The agents can, or can't communicate among them, their plans, their perceptions, and any other information that may be important to help the others agents' decisions. It will be used a blackboard to make the communication between agents.

The last element is the utility function used to allocate each agent to a mine (resource location). Many functions are possible. In this work the function utilized is the Euclidian distance from the mine to the Control Center, where the agents are supposed to lay down the gathered resource.

We have investigated two resource allocation centralized strategies (Auction and Elitist Social Welfare), and two decentralized ones (with and without communication), as described bellow:

- Auction Allocation: In this case, the central coordinator elects, based on the utility function, to which mine each agent will be allocated. The agent that has the best bid (best score in utility function) for a mine, is allocated to it.

- Elitist Social Welfare: This allocation score is equal to the best individual score from the agents. To get the best combination, each agent is allocated to the resource that has its best score, according to its utility function. If more than one agent have a preference for the same resource, the resource is allocated to the agent with best score.
- Individual without communication: Each agent, based on its utility function elects the best mining spot, and goes there. It has problems in the case another agent chooses the same mining spot.
- Individual with communication: Similar to the approach mentioned before, except that after each choice, the agent marks on a *BlackBoard,* how long time that mine will be occupied by it. If an agent picks an already occupied mine then it chooses the next option.

Implementation

All the simulations will happen in the same scenario (map), and will have the same game configuration. The evaluation criteria will be the amount of resource collected in the Control Center after the time has finished.

The reactive agent is based in [SIMONIN], the agents are going to create a potential field from the control center and spreading across all the map. During the creation of the potential field when an agent finds a mine it collects, and then go throw the gradient going to the lower values until it gets to the Control Center when it is closer enough to the Control Center it delivers the resource. While the agent is going to the Control Center he leaves "bread-crumbs" in the path he made it, so another agent that found those marks are going to follow it to the point with highest potential field value until it finds the resource.

Each cognitive agent has access to a class that is responsible to make his allocation to a mine(IAllocation), a class (GridManager) that grants it access to the map grid with mine positions and also responsible for path-finding. Those agents have a internal state machine with the states described below:

• IDLE: It's the initial state from this agent. In this state the agents is looking for a mining place, so he asks the IAllocation Class for one. When it gets his mining spot It asks for GridManager for a path to the spot, then it changes its internal state to GOING TO MINE.
• GOING TO MINE: In this state the agent follow the path that it has be given, until it finishes then sets its internal state to MINING.
• MINING: In this state the agent is going to mine the resource until the mine is empty or it gets full of resource. Then its state is set to GOING TO DELIVER.
• GOING TO DELIVER: In this state the agent is going to use the same path it got in IDLE state to get back to the Control Center, when it gets near the Control Center It delivers the resource that is carrying. Then its state is set to IDLE.

Above there's a diagram illustrating the cognitive agents state machine, and it's class diagram.

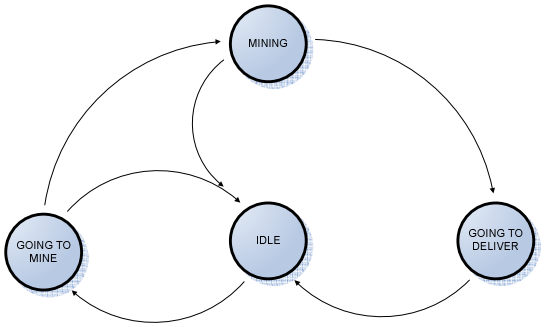


Figure 1 - Cognitive Agent State Machine

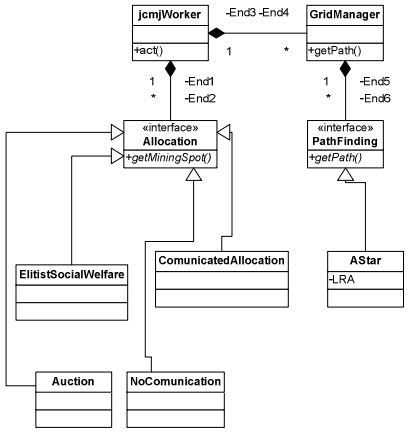


Figure 2 - Cognitive Agent Class Diagram

Scenario

The map (Figure 3) shows the scenario where the agents where simulated, during 800 cycles with 100 milliseconds each. The brown dots represents the resources, the gray dots are obstacles, the small blue dots are the workers and the big one is the Control Center.

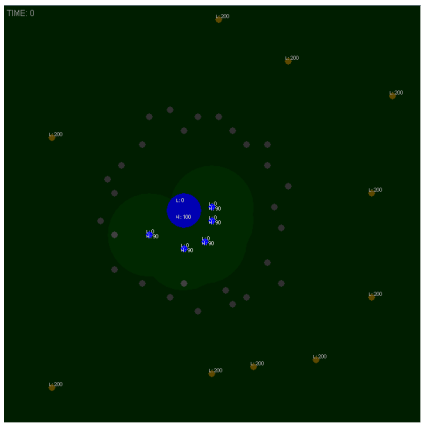


Figure 3 - Map

Results and Analysis

	Type	Coordination	Communication	Path-Finding	Allocation Strategy	Results
Worker0	Reactive		Yes	N/A	N/A	120
Worker1	Cognitive	Central	Yes	A-Star	Elitist	440
Worker2	Cognitive	Central	Yes	A-Star	Auction	120
Worker3	Cognitive	Individual	Yes	A-Star	Decentralized Communicated	340
Worker4	Cognitive	Individual	No	A-Star	Decentralized Without Communication	310
Worker5	Cognitive	Central	Yes	LRA	Elitist	550
Worker6	Cognitive	Central	Yes	LRA	Auction	490
Worker7	Cognitive	Individual	Yes	LRA	Decentralized Communicated	450
Worker8	Cognitive	Individual	No	LRA	Decentralized Without Communication	430

Table 1 - Summarizes the agents created with the combination of the different multi-agent architectures for resource gathering, and the results.

General Analysis

The reactive agents spent many time creating the gradient, but when an agent finds the mine it collects as much as it gets, and managed to get back to the control center, and back again to the resource very efficiently.
As it was expected the agents with the LRA Path finding got better results then its same version with the simple A-Star path-finding.
In decentralized allocation the agent has a performance increase when they could communicate, also the agents visits more different mines if compared with the agent without communication, where most of them usually choose the same resource to collect, decreasing the team performance.
In centralized allocation, the auction allocation got a worse performance due to the fact the closest mines got very crowded, and the agents had to keep recalculating their routes, when it was possible (LRA*). The Elitist Allocation got a better result because the agents got more spread, avoiding crowding a region.
Comparing the two types of coordination, the centralized got the best results, even when compared with the decentralized with communication.
The results shows that the best approach, among the analyzed ones, is the cognitive agent with centralized allocation based on Elitist Social Welfare and a path-finding that allows recalculating the route when it collides with another agent.

3. Conclusion

Despite the clear application of multi-agent gathering in games, particularly in RTS, no systematic work has been reported in the literature. In this paper, we have presented in this paper an original analysis of an ensemble of possible architectures or approaches for multi-agent resource gathering. In future works we would like to add more parameters to the architectures like different utility functions besides the distance used in this work.

Also we would like to expand the possibilities covered by the already existing parameters, as different approaches of collective path-finding and another communication strategy, besides the blackboard and flags used in this work, as directed messages. Another possibility is the creation of one allocation strategy with negotiation between the agents.

References

OSTEGAARD E.,SUKHATME G., MATARIC M., Emergent Bucker Brigading – A simple mechanism for improving performance in multi-robot constrained space foraging tasks.

SIMONIN O, Construction of Numerical Potential Fields with Reactive Agents.

SHELL D, MATARIC M, On foraging strategies for large-scale multi-robot system. *In:Proceedings of the 2006 IEEE/RSJ, Intl.Conference on Intelligent Robots and systems. Beijing, China October 2006*

HAYAT S, NIAZI M, Multi Agent Foraging- Taking a step further Q-Learning with Search. *In: Proceedings of the 2005 IEEE, International Conference on Emerging Technologies. Islamabad 2005.*

WANG S, TAN M, Relation between task-based diversity and efficiency in multi-robot foraging. *In: Control, Automation, Robotics and Vision, 2002. ICARCV 2002. 7th International Conference.*

STEELS, L. (1990). Cooperation between distributed agents through self organization. *In Demazeau, Y. and Müller, J.-P., editors, Decentralized AI — Proceedings of the First European Workshop on Modelling Autonomous Agents in Multi-Agent Worlds (MAAMAW- 89), pages 175–196. Elsevier Science Publishers B.V.: Amsterdam, The Netherlands.*

SUGAWARA K, WATANABE T, Swarming Robots – Foraging Behavior of Simple Multi-Robot System. *In:Proceedings of the 2002 IEEE/RSJ, Intl.Conference on Intelligent Robots and systems. EPFL, Lausanne, Switzerland October 2002.*

BOURG, D. AND SEEMANN, G.,. AI For Game Developers

SILVER, D.,. *Cooperative Path Finding*

Selecting Learning Algorithms for the Design of Virtual Players in Natural Resources Management Platforms

Guillaume MULLER and Jaime S. SICHMAN
Laboratório de Técnicas Inteligentes (LTI/PCS)
Escola Politécnica de Universidade de São Paulo
Av. Luciano Gualberto, 158, Trav. 3
05508-010 – São Paulo – SP – Brasil

Abstract

This paper focuses on the problem of designing Virtual Players for a particular type of serious game, in the domain of Natural Resources Management. Indeed, in some cases, it is impossible to play a particular game because there is not enough participants. Designing and implementing Virtual Players, that would stand in for some of the lacking human players, would allow to circumvent this limitation. In order to get realistic substitute players, it has been proposed to design the Virtual Players by learning actual human behaviours based on traces of games in which they are involved. However, there exists plenty of learning algorithms, with very different requirements and properties and no a priori best algorithm. This paper addresses more particularly the problem of selecting learning algorithms for the design of Virtual Players in Natural Resources Management "ABPS" platforms. Its main contributions are: a list of criteria for practically comparing the learning algorithms and the selection of some algorithms to be used in the design of Virtual Players for Natural Resources Management "ABPS" platforms.

Keywords:: Serious Games, Virtual Players, Learning Algorithms, Artificial Intelligence

Author's Contact:

guillaume.muller@freesurf.fr, jaime.sichman@poli.usp.br

Introduction

Serious Games are games whose ultimate goal is not only pure amusement. For some time now, researchers and teachers have shown a high interest in such games, since their entertaining aspect allows to raise or maintain the interest of the participants, even when addressing complex and serious problems. Such games have been used in various domains, like Natural Resources Management, as tools to support either training, multi-actor decisions, or social observation [Barreteau et al. 2001].

As a technical support for such games, a new kind of computer-mediated interaction platform has recently emerged. It is generally named "MAS/RPG" [Barreteau et al. 2001] or "ABPS" (Agent-Based Participatory Simulations) [Guyot and Honiden 2006], as it combines both Role Playing Games (RPG) and Multi-Agent-Based Simulation (MABS). It consists in making social actors play some roles in a simulation game, supported by (in the case of "MAS/RPG") or mediated by (in the case of "ABPS") a multi-agent platform.

Due to lack of availability, competence, interest, or to test repeatedly given configurations, it is sometimes impossible to group all the social actors at the same time to play all the mandatory roles of a particular simulation game. A solution to this problem is to design, implement and run Virtual Players, that would enact some of the roles needed by a particular simulation game. One of the challenges of the design of Virtual Players is to get realistic substitutes for the lacking human social actors, in order not to slant the results of the games. In order to get such realistic Virtual Players, it has been proposed to make the Virtual Players learn their behaviour directly from the ones of human social actors, based on traces of execution of the simulation games involving these Human Players.

However, there are plenty of learning algorithms in the literature, with very different requirements and properties, which makes them more or less suitable for the task. But, as WOLPERT's "No Free Lunch" theorem states, there is no a priori best learning algorithm. As a consequence, there is a need for a more practical methodology of selection of the algorithms, based on the particular application targeted. The general methodology of our work is to use the description of the application domain as a mean to extract a list of constraints on the Virtual Players, which is then translated into a list of properties that the requested learning algorithms must show. The application domain in our case is constituted by the "ABPS" platforms ViP-JogoMan [Adamatti 2007] and SimParc [Briot et al. 2008].

The results of this methodology and the main contributions of the paper are: (i) a list of criteria for a practical comparison of learning algorithms; and (ii) the selection of some algorithms to be used in the design of Virtual Players for "ABPS" platforms of Natural Resources Management.

Section 2 presents the criteria for comparing the algorithms, Section 3 details the comparison of the algorithms and Section 4 concludes and establishes directions for future works.

1 Constraints from the platforms

In this section, we identify some constraints that the functioning of the studied "ABPS" platforms [Adamatti 2007; Briot et al. 2008] imply on the choice of the learning algorithms used to design the Virtual Players.

1.1 Constraints for the category of algorithms

In this section we detail the characteristics of the platforms that will influence the choice of a particular category of algorithms.

- **Type of players** It is possible to differentiate three targets for the learning algorithms: (i) a Virtual Player to replace the Game Master (who plays special role in the process of decision-making); (ii) a Virtual Player for any other role; or (iii) a Virtual Assistant. The inputs and outputs of the players' decisions are different in each case: for instance, designing a normal player requires to integrates environmental information to make a decision, whereas designing a Game Master requires to handle partial decisions already made by other players.

- **Profiles for the roles** There can be different profiles for each role. For instance, a Game Master can have more environmental or more social tendencies. As a consequence, it is possible to distinguish two classes of learning problems: the first learning problem consists in grouping similar traces of behaviours for a same role into profiles; the second problem consists in learning to reproduce a particular profile.

- **Acquiring traces** Because the availability of the Human Players might be limited and the games take a long time to run, it is generally difficult to generate a large amount of traces. As a consequence, it is necessary: (i) to select games that will promote the production of relevant traces; and (ii) to select a category of learning algorithms able to rapidly extract important salient features.

1.2 Constraints for the internal model

In this section, we present the characteristics of the platforms that will influence the choice of the internal model of the players.

- **Granularity** The traces collected in the platforms will take the form of a sequence of actions (communicative or on the environment). The Virtual Players, however, can be trained and ran on pieces of information of different granularity levels: the whole sequence of actions, sub-sequences, or single decisions. It is particularly important to look at this criterion as the traces reveal the negotiation strategies of the players and a too small piece of information may make the learning algorithm unable to capture the link between actions, whereas a too large piece may make it link actions that should not.
- **Abstraction** The negotiations occurring in the platforms might involve several interactions referring to a same object and discussing some attributes of this object. For instance, during the negotiation consisting of a land owner buying a new area, the object of interest will be the land area and an attribute could be the price. It would therefore be interesting to learn to abstract the price variable, in order to learn the negotiation strategy, not the particular price proposed in this negotiation.
- **Human understandability** It can be important that the internal model is directly readable by humans, either for the designers to access the extracted features or to enable Virtual Players to explain their decisions (introspection). Other possibilities are that the internal model is easily translatable into an intelligible model, or that the model is questionable by the designers.

1.3 Constraints for the algorithm

The characteristics of platforms that will influence the final choice of the learning algorithms are presented in this section.

- **RPG rules and players' interactions** The rules of the RPG generally constrain the interactions between the players. There are two major elements that will influence the choice of the algorithm: (i) the way interactions occur, i.e. if there exists interaction protocols that the players must follow; and (ii) the communication language used, as it can be very structured or relatively free.
- **Stealth** It is important that the Virtual Players act like real Human Players. This means that they must be difficult to detect by Human Players. As a consequence, there is a need to choose a learning algorithm that makes Human-like decisions, not necessarily the most rational/logical ones.
- **Unpredictability** The behaviours of the Virtual Players should not be too predictable: even in similar configurations, they should not necessarily play exactly the same move. Of course, this should occur only when it is possible, i.e. the Virtual Players should not overstep the bounds of their profile nor of legal moves. This aspect is related to stealth as it is clear that both a very predictable and unpredictable Virtual Players will quickly appear to be artificial.
- **Innovation** It is not wanted that Virtual Players for Natural Resources Management platforms, learn to reproduce *exactly* what they observed in the traces. There is no notion of optimality or uniqueness of the solution to be reached by the learning algorithms. Also, Virtual Players will not be trained on every possible situation, they will face unknown situations. As a consequence, the learning algorithms should help explore new possibilities and innovate. This property is related to unpredictability and stealth, as Human Players generally have a prioris about machines not being innovative.
- **Adaptability** The players may evolve with time, as players gather more information in each time step, and with game repetitions, if the same players participate in several games. The Virtual Players could benefit adapting their behaviour dynamically. This step is related to their stealth, particularly if the same Human Players encounters the same Virtual Players in multiple games.

1.4 Constraints on the inputs

This section presents the constraints that the characteristics of platforms will imply on the inputs of the learning algorithms.

- **Similarity** Whatever the use of learning algorithms: to identify profiles or reproduce a profile, it is necessary to define a similarity measure between two traces. In the identification task, the similarity measure is used to compare two traces to decide to group or separate them, whereas in the reproduction task, it is needed to evaluate the performance of the learning process, by comparing the output traces, generated by the algorithm, to the input traces, initially generated by the Human Players.

2 Criteria for a practical comparison

In this section, we briefly present the criteria, extracted from the application domain, that will enable us, in the next section, to compare the (classes of) learning algorithms.

The following items presents these criteria, the links with the constraints of the previous section and the possible values for each criterion:

- **Category**: Due to the type of decision to be made, there is a direct relation between the *Type of Player* (Section 1.1) and the *Category* of the learning algorithm to be selected. Algorithms are compared based on the category they belong to. This criterion admits six possible values: *Supervised*, *Unsupervised*, *Semi-supervised*, *Reinforcement*, *Transduction*, and *Multi-task*.
- **Aim**: Section 1.1 (*profiles*) shows that there are two learning problems: classify the traces into profiles (without necessarily having the labels), and learning to reproduce a labelled profile. Algorithms are compared based on the problems they can address. This criterion admits two possible values: *classification* (supervised or not) into profiles; and *reproduction*.
- **Inputs/Outputs**: The learning algorithms are compared according to the *granularity* of the inputs and outputs that they can handle (Section 1.2). This criterion admits four values, combinations of: *single*, if the algorithm is able to handle single players' decisions; and *sequence*, if the algorithm is able to handle (sub-)sequences.
- **Number of examples**: In our application domain, there are generally *few traces* (Section 1.1). The algorithms are compared according to the quantity of examples needed. This criterion will admit three possible values: *many* examples, if the algorithm needs many examples to learn correctly; *few* examples, if it is able to produce correct results even with few examples, and *normal* if it does not fall in the previous categories.
- **Generalisation**: the learning algorithms are compared based on their ability to abstract or not the examples learnt. Generalisation helps cope with *abstraction* (Section 1.2), *innovation*/unknown situations and *stealth* (Section 1.3). This criterion admits a binary response: *yes*, if it is able to generalise or *no*, if it is not.
- **Internal model**: The learning algorithms are compared based on the *internal models* they accept (Section 1.2) This criterion admits two possible values: *numerical*, for learning algorithms that can accept an internal model like statistics; and *symbolic*, for learning algorithms that accepts an internal model like logics.
- **Intelligibility**: the learning algorithms are compared based on their ability produce *intelligible* models (Section 1.2). This criterion admits three values: *direct*, if it is directly intelligible; *translatable*, if it is translatable into an intelligible model;

and *questionable*, if it can be kept as a questionable "black-box".

- **Non-determinism**: The algorithms are compared along their ability to produce output that is not completely deterministic. This helps for *unpredictability* (Section 1.3), *innovation* and *stealth* (Section 1.3). This criterion admits a binary response: *yes*, if it is able to produce output that is not necessarily completely deterministic or *no*, if it is not.
- **Incremental**: the learning algorithms are compared based on their ability to cope incrementally with new examples. This helps for *adaptation* (Section 1.3) and to run on-line Virtual Assistants (Section 1.1). This criterion admits two possible values: *yes*, if the algorithm is working in an incremental way; and *no*, if it is not.
- **A priori knowledge**: The learning algorithms are compared judging their ability to accept *a priori knowledge* (e.g., heuristics) or not (Section 1.3). This criterion admits three possible values: *cannot*, if it cannot accept a priori knowledge; *need*, if it needs a priori knowledge to be able to function; and *can*, if it can accept a priori knowledge, but do not necessarily need it to work.

3 Practical comparison of the learning algorithms

As there exists many algorithms in the literature, we have presented in the table only large classes of learning algorithms. The selection of the classes is based on the functional approach exploited in [Mitchell 1997], which discerns nine classes of algorithms based on their functioning: Decision tree, Neural Networks, Bayesian, Instance-based, Genetic, Set of Rules, Analytical, Reinforcement, and SVM. The filling of the table has been made considering a few representatives for each class: for **Decision tree**, we have used ID3 and C4.5; for **Neural Networks**, we have used the perceptron and multi-layered neural networks; for **Bayesian**, we have used Naive Bayes and Expectation-Maximisation (EM); for **Instance-based**, we have used Case-Based Reasoning (CBR) and k-Nearest Neighbours (kNN); for **Genetic**, we have used the general concepts, plus the Strongly-Typed Genetic Programming (STGP) [Montana 1995]; for **Set of Rules**, we have used RIPPER, AQ and CN2; for **Analytical**, we have used Prolog-EBG; for **Reinforcement**, we have used Q-Learning and Temporal differences plus some Inverse Reinforcement Learning [Abbeel and Ng 2004; Ng and Russell 2000]; for **SVM**, we have used general concepts.

Table 1 presents, in a concise way, the results of our comparison of these various classes of learning algorithms. A legend of the abbreviations and symbols used is present in the last row.

The values for the first column have been filled based on information from several classical works on learning algorithms [Kotsiantis 2007; Alpaydın 2004; Mitchell 1997]. The line about Analytical Learning is mainly based on [Knuutila 1999] , the line about Genetic Programming on [Koza 1992] and the line about Reinforcement Learning on [Mitchell 1997].

In the "Aim" columns, the ability to classify has been mainly established thinking about a typical task of supervised classification. The ability to reproduce is based on a synthesis of the four next columns: a check-mark is put where the four next columns have a check-mark and a tilde is put as soon as there is more than one tilde in the next columns.

In order to fill the "In/Out" columns, we have considered the task of reproduction has a supervised classification task: the algorithm is trained to associate a decision or sub-sequence of decisions with the reacting decision or sub-sequence of decisions; and the algorithm is used by being provided the current decision or sub-sequence of decisions occurring in the game and asked to provide the next move (single decision) or moves (sub-sequence of decisions). The tildes 1–4, 5–7, 10–12, represent the fact that it can be hard to represent an arbitrary length sequence as the label for Neural Networks, Instance-based, and SVM algorithms.

The column for "number of examples" has been filled mainly based on [Kotsiantis 2007], where *normal* is put when the paper does not explicit the need for few or many examples. The *not relevant* marks have been put for Genetic Programming and Reinforcement Learning because they do not naturally work based on a set of training examples like the other algorithms, but respectively on size and iteration over a population and reinforcement signal.

In order to fill the "Generalisation" column, we have considered the task of responding to unknown situations. The tildes 13, 14, 16, 17 are there to underline potential problems with over-fitting. However, there are simple solution to prevent this, for instance pruning for Decision Trees and Sets of Rules, so these kinds of algorithms have not been totally rejected.

The "Internal model" values have been filled based on information from the same classical works on learning algorithms than column one. The coma separator signifies that both models can be used, whereas the pipe (|) signifies that there are models that do use rather symbolic representation (e.g., Case-Based Reasoning) whereas others use rather numerical internal models (e.g. k-Nearest Neighbours).

Generally, symbolic are more intelligible then numerical models. *questionable* has been put when [Kotsiantis 2007] criticises the algorithm's intelligibility and *direct* when it says it is good. *translatable* is used for models like Bayesian Networks and Reinforcement Learning, which can be understood with a little practise or converted into a more human-readable model

The non-determinism has been judged based on the ability of each learning algorithm to produce non-deterministic models. For instance, to produce probabilistic models that say: "in situation S, do action X with 80% probability or action Y, with 20% probability".

The incremental property is related with the ability to run the algorithm on-line. To judge this aspect, we have considered the very functioning of the algorithm, i.e. whether the algorithm only revises a small part of its model when a new example is presented or if it needs to rebuild everything from scratch. According to [Kotsiantis 2007], Decision Trees, Sets of Rules and SVM are very bad at managing new examples. [Knuutila 1999] says that Prolog-EBG has a sequential covering. Neural Networks and Reinforcement Learning are very adapted to such conditions.

Concerning the last column, according to [Knuutila 1999] (resp. [Mitchell 1997]), Analytical Learning (resp. Bayesian Networks) requires a domain theory (resp. bootstrap information). With very simple algorithms like Q-Learning, Reinforcement Learning does not need a priori knowledge. However, utility-based algorithms require a significant amount of it. That is why we used the pipe notation. Finally, *can* has been used when heuristics can be implemented to help the algorithm learn more precisely or more quickly.

4 Possibilities

In this section, we discuss the possibilities regarding our needs and the characteristics of the learning algorithms.

By combining the two first columns of Table 1, our choice for the unsupervised classification problem is reduced to: Neural Networks, Bayesian Networks, and Instance-based learning. By adding that number of examples and ability to generalise are the more important criteria for our problem and that a priori knowledge is an interesting property, we can remove the first option and keep as possibilities Bayesian Networks (which is better because of few examples) and Instance-based learning (which is better as it accepts heuristics).

Concerning the reproduction problem, the criteria that are very important are: number of examples; ability to handle sequences (In/Out and internal model); ability to generate non-deterministic behaviours; and ability to generalise. Based on these criteria, only remain the Bayesian Networks, Genetic Programming and Reinforcement Learning possibilities. Bayesian Networks is rejected as it suffers from the assumption of independence of among child

	Category	Aim		In/Out				# Examples
		class	repro	dec/dec	dec/seq	seq/dec	seq/seq	
Decision tree	super	✓	✓	✓	✓	✓	✓	normal
Neural Networks	super,unsup^{al},multi^{al}	✓	$\sim_1$	✓	$\sim_2$	$\sim_3$	$\sim_4$	many
Bayesian	super,unsup^{al},semisup^{al}	✓	✓	✓	✓	✓	✓	few
Instance-based	super,unsup	✓	$\sim_5$	✓	$\sim_6$	✓	$\sim_7$	normal
Genetic	reinf^{al},multi^{al}	✓	✓	✓	✓	✓	✓	$-_8$
Set of Rules	super	✓	✓	✓	✓	✓	✓	normal
Analytical	super	✓	✓	✓	✓	✓	✓	few
Reinforcement	reinf	✓	✓	✓	✓	✓	✓	$-_9$
SVM	super	✓	$\sim_{10}$	✓	$\sim_{11}$	✓	$\sim_{12}$	many
Legend	**S**upport **V**ector **M**achine; **super**vised; **unsup**ervised; **semi-sup**ervised; **reinf**orcement; **multi**task. **class**ification; **repro**duction. ✓ = yes; ✗ = not possible without modifying the algorithm; $\sim$ = not immediate, but possible. singl **dec**ision; **seq**uence of decisions. – = not relevant.							

	Generalization	Int. Model	Intelligibility	Non-determinism	Incrementality	A priori
Decision tree	$\sim_{13}$	sym	direct	✗	✗	can
Neural Networks	$\sim_{14}$	num	quest	✓	✓	cannot
Bayesian	✓	num	transl	✓	✓	need
Instance-based	✓	sym^{al} |num	quest	✓	✓	can
Genetic	✓	sym,num	$-_{15}$	✓	✗	can
Set of Rules	$\sim_{16}$	sym	direct	✗	✗	can
Analytical	✓	sym	direct	✗	✓	need
Reinforcement	✓	num	transl	✓	✓	can|need
SVM	$\sim_{17}$	num	quest	✗	✗	can
Legend	✓ = yes; ✗ = not possible without modifying the algorithm; $\sim$ = not immediate, but possible. **num**eric; **sym**bolic. **transl**atable; **quest**ionable. – = not relevant. $_x$: referred in explanations below.					

Table 1: *Practical comparison of general learning algorithms classes.*

nodes. The advantage of Genetic Programming is that the internal model can be chosen. With reinforcement learning the Virtual Player can adapt dynamically.

Conclusion and future work

In this paper, we address the problem of designing Virtual Players for a particular type of serious games, the "ABPS" platforms, in the domain of Natural Resources Management. As the Virtual Players must show a credible behaviour, it was proposed to make them learn their behaviour based on traces of execution of games involving the Human Players they should replace. In order to select the learning algorithms to perform such a task, we extracted a list of criteria from the functioning of the platforms, in order to make a practical comparison of the learning algorithms and proposed some learning algorithms that appear more adapted to our goal.

There are two main contributions of this paper: the methodology to choose criteria for practical selection of the learning algorithms and the selection of some algorithms to be used.

Practical future work consists in implementing and evaluating the Virtual Players. This will be possible only when a working platform will be publicly available, i.e. (i) *runnable*, in order to actually collect traces and (ii) *modifiable*, in order to insert the Virtual Players. A more theoretical future work will consist in generalising the methodology developed here, in order to propose algorithm selection criteria for other type of Serious Games.

Acknowledgements

Dr. MULLER is supported by CNPq grant 382737/2008-3.

References

ABBEEL, P., AND NG, A. Y. 2004. Apprenticeship learning via inverse reinforcement learning. In International Conference on Machine Learning (ICML'04), ACM Press.

ADAMATTI, D. F. 2007. Inserção de Jogadores Virtuais em Jogos de Papéis para Uso em Sistemas de Apoio a Decisão em Grupo: um Experimento no Domínio da Gestão de Recursos Naturais. PhD thesis, Universidade de São Paulo, São Paulo, Brasil.

ALPAYDIN, E. 2004. Introduction to Machine Learning (Adaptive Computation and Machine Learning). MIT Press.

BARRETEAU, O., BOUSQUET, F., AND ATTONATY, J. 2001. Role-playing games for opening the black box of multi-agent systems: method and lessons of its application to senegal river valley irrigated systems. Journal of Artificial Societies and Social Simulation (JASSS) 4, 2.

BRIOT, J.-P., VASCONCELOS, E., ADAMATTI, D., SEBBA PATTO, V., IRVING, M., BARBOSA, S., FURTADO, V., AND LUCENA, C. 2008. A computer-based support for participatory management of protected areas: The simparc project. In Seminário Integrado de Software e Hardware (SEMISH'08), Sociedade Brasileira de Computação (SBC), Brazil, Belém, PA, Brazil, 1–15.

GUYOT, P., AND HONIDEN, S. 2006. Agent-based participatory simulations: Merging multi-agent systems and role-playing games. Journal of Artificial Societies and Social Simulation (JASSS) 9, 4, 8.

KNUUTILA, T., 1999. Machine learning lectures, chapter 11: Analytical learning. Built in: 1999. Last update: May 24, 2004. Accessed July 2009. `http://guardian.cs.utu.fi/knuutila/Courses/ML/Default.htm`.

KOTSIANTIS, S. 2007. Supervised machine learning: a review of classification techniques. Informatica (October).

KOZA, J. R. 1992. Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press.

MITCHELL, T. 1997. Machine Learning. McGraw Hill.

MONTANA, D. J. 1995. Strongly typed genetic programming. Evolutionary Computation 3, 2, 199–230.

NG, A. Y., AND RUSSELL, S. 2000. Algorithms for inverse reinforcement learning. In International Conference on Machine Learning (ICML'00), Morgan Kaufmann, 663–670.

Stimulating imitation of children with Down syndrome using a game approach

André Brandão[1], Erick Passos[1], Cristina Vasconcelos[2], Aura Conci[1], Esteban Clua[1], Silvia Brandão, Pedro Thiago Mourão[1], Marcos Cordeiro d'Ornellas[3]

[1]MediaLab – UFF – Universidade Federal Fluminense, Niterói, RJ, Brasil
[2]PUC-Rio – Pontifícia Universidade Católica do Rio de Janeiro, RJ, Brasil
[3]LaCA – UFSM – Universidade Federal de Santa Maria, RS, Brasil

Abstract

The usage of computer games by people with special needs sometimes can be difficult because some of them are not used to interact with a keyboard, mouse or gamepads. Some computer games even offer different paradigms of interaction but are no specifically designed to help people with special needs. We are currently facing the problem of developing a game to help Down syndrome children to interact with computers by the means of imitation. Our work presents the first step to provide an interaction framework to automatically recognize broad movements using computer vision concepts for movement recognition.

Keywords: movement recognition, people with special needs, Down syndrome, children

Authors' contact:
andrebrandao@ic.uff.br
epassos@ic.uff.br
crisnv@gmail.com
aconci@ic.uff.br
esteban@ic.uff.br
silviabrandao7@hotmail.com
pedrothiago@hotmail.com
marcosdornellas@gmail.com

1. Introduction

Recent works in Computer Science are showing that people with cognitive impairment have been using more computer applications in the last years [Feng et al, 2008]. Some of these studies pay a special attention to people that have special needs such as autism [Veeraraghavan And Srinivasan, 2007] and Down syndrome [Feng et al, 2008; Marti, 2009]. However, even presenting promising results based on experiences with people with Down syndrome, the authors did not find any game that was developed specifically for this kind of user.

Some studies presents results that demonstrates that people with a special kind of cognitive impairment, in our case, Down syndrome, can use mouse with difficulties but is very hard to other forms of interaction, like keyboard and gamepads [Feng et al, 2008]. Down syndrome affects an individual's overall development, including areas such as cognition, sensory perception and processing, gross and fine motor skills, and also short term memory [Abbeduto, 2001; Kumin, 2003].

At the same time, speech therapists have been trying to stimulate users with impairments by observing their interaction with games. However, these games are not developed for people with such impairments. Specifically, we have not found any game developed for children between 3 to 5 years old with Down syndrome. We designed the JECRIPE game to fit this niche, and in this paper we describe some challenges that need to be tackled.

Our proposal is to offer a different paradigm of interaction between users and games that applies movement recognition based on Computer Vision techniques. The goal of this work is to develop algorithms to extract characteristics from frames of video in real-time with the use of Graphics Processing Unit (GPU's).

This paper is organized in three more sections. In section 2, we describe some related works about interaction of Down syndrome children and movement recognition. Section 3 presents some movement recognition concepts and some highlights of our game. Finally, in section 4 we expose some concluding remarks and future work.

2. Related Work

We decided to divide this section in two parts: in this first we compare our work with some research on the interaction of children with Down syndrome and computer games; the second part will present some related work on movement recognition based on computer vision algorithms.

2.1. Users with Down syndrome

There are some works with users with impairment; some of them are about autism [Marti, 2009] and others in our context: Down syndrome. We highlight two of those works: Computer Usage by Young Individuals with Down syndrome [Feng et al, 2008] and Creative Interactive Play for Disabled Children [Marti, 2009].

The first work [Feng et al, 2008] presents experiences with people between 4 and 21 years old with Down syndrome. It was made a survey asking how people with Down syndrome make use of computers. In specific questions about input device, the most cited were keyboard (85.6%) and mouse (93.2%). Other input devices cited were: touch screen (12.3%), joystick (7.5%), touchpad (5.5%), trackball (4.9%), speech recognition (3.4%), stylus (2.3%) and keyguard (0.4%). One of the conclusions of the work was people with Down syndrome have much difficult to use mouse and specially keyboard because they have fingers shorter than usual.

Other work [Marti, 2009] exposes different manners of interaction of handicapped children with computers. One kind of interaction mentioned was a remote controlled robot used by a seven year old boy with autism. Other robots were used in experiences, some of them, social robots. The objective of the work [Marti, 2009] was to collect and discuss research from robotics to interactive environments and tangible media promoting play for physically, visually and hearing for different handicapped children.

Those works showed important concepts and results for our work, but is difficult to find a great number of papers that use experience with handicapped children, especially with Down syndrome children, interacting with computers [Feng et al, 2008]. Based on results of [Feng et al, 2008] it's perceptible that is necessary to make different manners of interaction. Our objective is not the same as [Marti, 2009] because we want to take care of only children with Down syndrome at this moment. In the next section we describe how the movement recognition will work in our search.

2.2. Movement Recognition

Several types of motion capture systems are presented in the literature and some of them were instrument of interest in our work. The types of motion capture that we were interested are: Marker-based motion-capture, Color markers and Bare-hand tracking.

Marker-based motion-capture: these systems require obtrusive retro-reflective markers and many camera setups [Park, and Yoon, 2006]. Our pretension is to make use of one single camera to make less expensive for users, so we discarded this option.

Color markers: the usage of color markers are used in [Wang and Popović, 2009] and presented good results. The work introduced a special glove design consisted in large color patches accounts for camera limitations to identify the movements. Our approach has no intended to make use of gloves because our proposition is to identify movements of the body, not only color marked regions.

Bare-hand tracking: edge detection and silhouettes are the most common features used to identify the pose of the hand [Wang and Popović, 2009]. Our approach is intended to make use of this type of motion capture technique but we are going to make use of skeletonization applied in skin segmentation. The work [Stenger et al, 2006] shows a Model-based hand tracking using a hierarchical Bayesian filter. It presents a set of contributions in this type of tracking like: a hierarchical filtering algorithm, which combines robust detection with temporal filtering and the formulation of likelihood function that fuses shape and color information, significantly improving the robustness of the tracker.

However the work [Stenger et al, 2006] exposes a good model, it is restricted only for hand movements. Our objective is to identify not only hand movement but either other parts of the body like head and neck. That's why were are going to use a skin segmentation technique and it is in development to present results in real-time, implemented in GPU, making use of a filter and thinning to handle body movements.

The next section presents our JECRIPE game which has the pretension to stimulate children with trisomy 21.

3. The JECRIPE game

Our game, named JECRIPE, is an approach to stimulate children with Down syndrome and in pre-scholar age. Studies have demonstrated that a few areas of cognition need to be stimulated to help the development of the kind of children that our game targets [Brandao, 2006]: **Imitation**, Perception, Fine Motricity and Visio-Motor Integration, Comprehension and Expressive Language. The JECRIPE game is currently in development and this work highlights the challenges in the development of the Imitation task of it.

Imitation has a fundamental relation with language acquisition [Schopler Et Al, 1990]. Learning words is part of the language acquisition and imitation helps in this process. Motor imitation must be learned before the language stimulation [Brandao, 2006]. Imitation also plays an important role in socialization, because through it a child learns how to behave, cooperate and respond [Brandao, 2006]. However, it is important to observe that the imitation capacity of children with Down syndrome is significantly lower than those without it [Guerrero López, 1997].

Figure 1: Illustration of an open arms movement.

Children with trisomy 21 (Down syndrome – DS) have the tendency to favor the use of gestural communication instead of spoken language, also avoiding combining vocalizations with hand drawings [Smith, 1987]. Another work [Tristão, and Feitosa, 1998], exposes the effectiveness of imitation on the verbal behavior of children with impairments, concluding that imitation plays an important role in language acquisition.

In view of the results of the mentioned works, the imitation will be addressed in JECRIPE game through the display of choreographed movements with music of public domain. In the Brazilian culture, there are some songs with choreography that exercises some movements. These movements include pointing parts of the body (hands, fingers, eyes, nose, mouth, ears, arms, hair), open the arms, and shake the body following the music's rhythm. Figure 1 and 2 illustrate some of these movements.

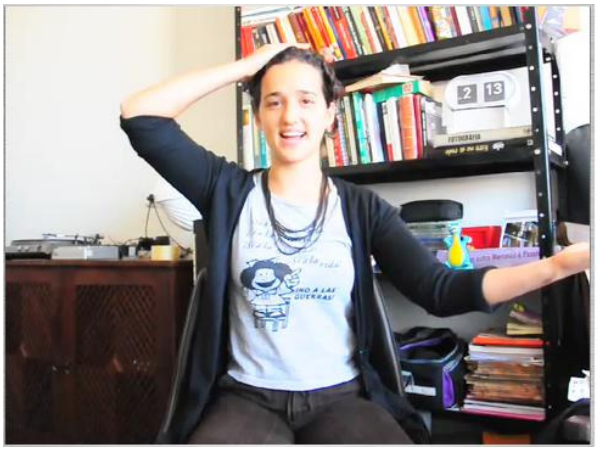


Figure 2: Illustration of a pointing movement.

In the game, this choreography is performed by a character with DS resemblance (Figure 3). The choice of using DS features in a character is justified because it facilitates the empathy between the user and the game character. There are no known games that use a DS character to date.

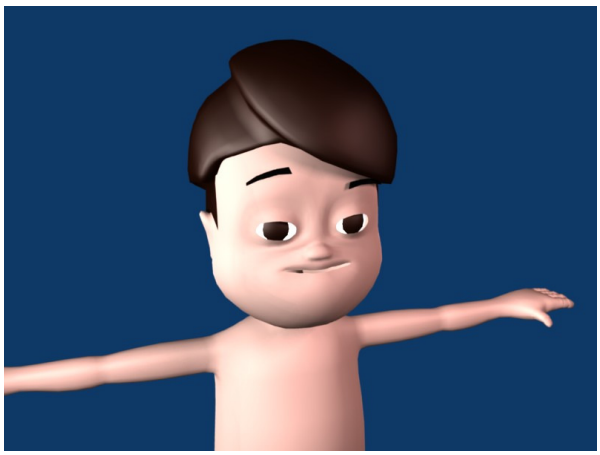


Figure 3: Character with Down syndrome characteristics.

JECRIPE players have to mimic the character, which is modeled in 3D and automatically choreographs the songs. The movement recognition module of the game is responsible to recognize if the movements are correct or not. Some of the choreographies are illustrated in Figures 4 and 5.

4. Expected Results

In this paper, we exposed some concepts of a game to stimulate children with DS. Our next steps are to conclude the development of the JECRIPE game, specially the movement recognition module mentioned in this text. Our goal is to offer an easy interaction for users with impairment, by recognizing broad gestures in front of a single camera. This kind of interaction may facilitate the interaction of this kind of user in comparison to a keyboard, mouse or gamepad.

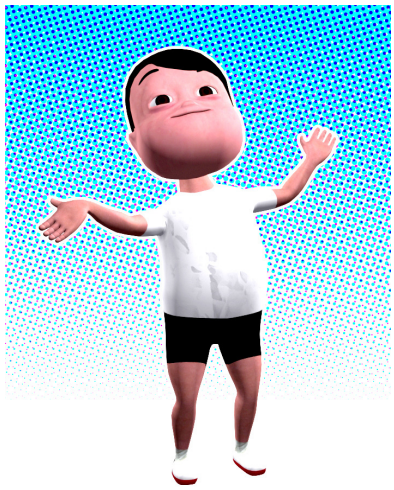


Figure 4: Open arms movement by the character

We are implementing the necessary Computer Vision algorithms for movement recognition in CUDA [ref]. We plan to adapt or extend the model-based hand tracking using a hierarchical bayesian filter presented

in [Stenger et al, 2006] for our needs. This extension is targeted to recognize broad body movements, instead of hand tracking only.

Finally, we expect to make experiments with children with Down syndrome in pre-school age (3 to 5 years old) to assure that this interaction paradigm is interesting for this kind of user. We believe that our work promises contributions to the Human-Computer Interaction and Computer Vision areas, at the same time helping people with special needs by the use of computer games.

Figure 5: Pointing movement by the character

References

ABBEDUTO, L., PAVETTO, M., KESIN, E., WEISSMAN, M., KARADOTTIR, S.,O'BRIEN, A. & CAWTHON, S., 2001, "THE LINGUISTIC AND COGNITIVE PROFILE OF DOWN SYNDROME: EVIDENCE FROM A COMPARISON WITH FRAGILE X SYNDROME". DOWN SYNDROME RESEARCH AND PRACTICE,7, 9-15.

BRANDAO, S. R. S., 2006. DESEMPENHO NA LINGUAGEM RECEPTIVA E EXPRESSIVA DE CRIANÇAS COM SÍNDROME DE DOWN. MÁSTER DISSERTATION, FEDERAL UNIVERSITY OF SANTA MARIA.

FENG, J. AND LAZAR, J. AND KUMIN, L. AND OZOK, A., 2008, "COMPUTER USAGE BY YOUNG INDIVIDUALS WITH DOWN SYNDROME: AN EXPLORATORY STUDY" ASSETS '08: PROCEEDINGS OF THE 10TH INTERNATIONAL ACM SIGACCESS CONFERENCE ON COMPUTERS AND ACCESSIBILITY PAGES 35-42 HALIFAX, NOVA SCOTIA, CANADA, ACM.

GUERRERO LÓPEZ, J. F. NUEVAS, 1997 PERSPECTIVES EN LA EDUCACIÓN E INTEGRACIÓN DE LOS NIÑOS CON SÍNDROME DE DOWN. PAIDÓS, BARCELONA.

KUMIN, L., 2003, "EARLY COMUNICATION SKILLS IN CHILDREN WITH DOWN SYNDROME: A GUIDE FOR PARENTS AND PROFESSIONALS". BETHESDA, MD: WOODBINE HOUSE.

MARTI, P. AND POLLINI, A. AND RULLO, A. AND GIUSTI, L. AND GRONVALL, E., 2009, "CREATIVE INTERACTIVE PLAY FOR DISABLED CHILDREN" IDC '09: PROCEEDINGS OF THE 8TH INTERNATIONAL CONFERENCE ON INTERACTION DESIGN AND CHILDREN PAGES 313-316 ACM, NEW YORK, NY.

PARK, J., AND YOON, Y. 2006. LED-GLOVE BASED INTERACTIONS IN MULTI-MODAL DISPLAYS FOR TELECONFERENCING. IN INTERNATIONAL CONFERENCE ON ARTIFICIAL REALITY AND TELEXISTENCE–WORKSHOPS (ICAT), 395–399.

SCHOPLER, E.; REICHLER, R.J.; BASHFORD, A.; LANCING, M.D.; MARCUS, L.M., 1990, PSYCHOEDUCATIONAL PROFILE REVISED (PEP-R). TEXAS: PRO-ED.

SMITH, L., 1987, THE STRUCTURE OF DIALOGUE IN EARLY LANGUAGE DEVELOPMENT: LONGITUDINAL CASE STUDIES OF DOWN SYNDROME AND NONRETARDED TODDLERS. IN: RAUTH, H. & H.C. STEINHAUSEN H.C. (ORGS), PSYCHOLOGY AND EARLY DEVELOPMENT, 201-213, NORTH HOLLAND: ELSEVIER,.

STENGER, B., THAYANANTHAN, A., TORR, P., AND CIPOLLA, R. 2006. MODEL-BASED HAND TRACKING USING A HIERARCHICAL BAYESIAN FILTER. IEEE TRANSACTIONS PATTERN ANALYSIS AND MACHINE INTELLIGENCE 28, 9, 1372–1384.

TRISTÃO, R. M.; FEITOSA, M.A., 1998, LINGUAGEM NA SÍNDROME DE DOWN. PSICOLOGIA: TEORIA E PESQUISA, 14 (2)127-137.

VEERARAGHAVAN, S. AND SRINIVASAN, K., 2007, "EXPLORATION OF AUTISM USING EXPERT SYSTEMS" INFORMATION TECHNOLOGY. ITNG '07. FOURTH INTERNATIONAL CONFERENCE ON 2-4 APRIL 2007 PAGE(S):261 - 264.

WANG, R. Y. AND POPOVIĆ, J. 2009 REAL-TIME HAND-TRACKING WITH A COLOR GLOVE ACM TRANSACTIONS ON GRAPHICS 28(3).

The NX iPhone 2D Gaming Framework

Eduardo Coelho
George Ruberti Piva
Nexia Mobile Solutions

Paulo César Rodacki Gomes
Dalton Solano dos Reis
Universidade Regional de Blumenau (FURB)

Abstract

The recent release of Apple's iPhone SDK opened new possibilities for mobile game development. Currently available commercial and open-source game engines still lack support for some specific iPhone features such as NIB files, UIKit and Bonjour. This paper presents NX 2D Gaming Framework, a framework for rapid 2D game development with better integration to such specific iPhone technologies. With NX 2D Gaming Framework, developers might integrate OpenGL and UIKit interfaces in single-player and multi-player iPhone games over Wi-Fi networks using Zero Configuration Networking Standard and multi-touch interface.

Keywords:: iPhone, Mobile Game Engine, Zero-configuration

Author's Contact:

{eduardo,piva}@nexiamobile.com
{rodacki,dalton}@inf.furb.br

1 Introduction

The recent release of Apple's iPhone SDK opened new possibilities for mobile game development. iPhone has some unique features among existing mobile platforms such as its programming language, development environment, device's hardware resources, operating system and even the business model for application distribution.

This paper presents NX 2D Gaming Framework, a lightweight and compact 2D game development framework based on Apple's iPhone SDK. It's main purpose is provide developers a set of tools for rapid development of small-scale 2D game projects.

The main differences among this framework and other available engines lies in its integration with features that are exclusive to iPhone. The NX 2D Gaming Framework allows simultaneous utilization of OpenGL ES and Apple's UIKit framework. Also, it provides support for NIB files[1] and specifies a communication protocol for multi-player games over Wi-Fi networks based on Apple's Bonjour technology. In the following sections the NX 2D Gaming Framework will be referred simply as NX Framework.

Beyond this introduction, this paper is divided into 5 more sections. In section 2 some related work are presented. Section 3 shows the iPhone OS technologies, including hardware issues and operating system. Section 4 details the NX Framework architecture and its underlying technologies. For a more concrete examination and validation of results, the section 5 shows two game projects based on NX Framework. Finally, some results and future work are cited in section 6.

2 Related Work

Some related works which also aims to make the game development for the iPhone platform easier were identified, these include the open-source frameworks SIO2 [SIO2 2009], Cocos2D-iPhone [cocos2d 2009] and Oolong [Engel 2009], the commercial game engines Unity [Unity Technologies 2009], iTGB [GarageGames 2009] and Ston3D [Stonetrip 2009]. Few academic projects focusing in Apple's iPhone SDK have been published until the present date. Comparing to NX 2D Gaming Framework the cited game engines still have characteristics that are too much adherent to game development for desktop platforms.

Most of these engines provide support for iPhone specific features such as accelerometers and multi-touch screen, however they all lack support for specific technologies such as Apple's Interface Builder files, iPhone's UIKit framework and Bonjour (Apple's implementation of Zero Configuration Networking Standard). The use of those technologies brings some advantages such as better integration with the device's operating system services and user interface system, faster code development and less probability of error occurrence.

3 iPhone OS Technologies

The iPhone 3G is a smartphone powered by an ARM 620 Mhz processor with 128 MB RAM memory, 8 and 16 GB flash drive. The recent released iPhone 3G S upgrades to 16 and 32 Gb flash drive. Moreover, this device comes with proximity and ambient light sensor, multi-touch 320×480 screen, 3-axis accelerometer and magnetometer.

The iPhone OS, which is the iPhone operating system, is based on a variant of the same Darwin operating system core that is found in Mac OS X [Allen and Appelcline 2008]. Regarding the software development for this platform, Apple has released the iPhone SDK on June 2008, which includes an Objective-C compiler and IDE (Xcode), iPhone simulator, and a suite of additional tools for developing iPhone and iPod applications [Dalrymple and Knaster 2008].

The iPhone SDK provides a rich set of APIs that are usefull for game development, including multi-touch event and accelerometer support, 2D and 3D rendering with OpenGL ES 2.0, audio playing back; network communication infrastructure, data persistence, views and windows abstractions, hardware assisted animation support and so forth. The frameworks av ailable for developers are found in the iPhone OS [Apple Computer 2009a], which can be viewed as a set of abstraction layers, as depicted in figure 1.

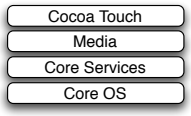


Figure 1: *iPhone OS layers*

At the lower layers of the system are the fundamental services on which all applications rely, while higher-level layers contain more sophisticated services and technologies. Higher-level layers provide object-oriented abstractions for lower-level constructs, but not necessarily mask the technologies contained in the lower layers.

The Cocoa Touch Layer comprises the UIKit and Foundation frameworks, which provide the basic tools and infrastructure necessary to implement graphical, event-driven applications in iPhone OS. The Media layer provides the key technologies to 2D and 3D drawing and audio playbing back. The most notable frameworks present on this layer are: OpenGL ES, QuartzCore, Core-Graphics, AudioToolbox, OpenAL and Media Player. The Core Services layer provides the fundamental system services that are used by all applications, such as collection data types, String date and time. Finally, the Core OS layer encompasses the kernel environment, drivers, and basic interfaces of the operating system.

[1] A NIB file describes applications user interfaces that were previously constructed in the Apple's Interface Builder WYSIWYG editor and are based on Apple's UIKit framework.

4 The NX iPhone 2D Gaming Framework

The iPhone hardware capabilities and the rich set of APIs provided by the iPhone SDK are keen on for gaming development. The NX Framework appears as an abstraction for those technologies, which makes the game development for this platform easier. By using the framework, the developer is able to take advantage of its abstraction while being able to access lower levels of iPhone OS' APIs as well.

The NX Framework is written in the Objective-C programming language and so was modeled under the object-oriented programming paradigm. The Objective-C language is quite interesting because it has a memory management system (garbage collection). Furthermore, it is compatible with C and C++ programming languages, giving to the developer a lot of flexibility concerning the technologies to be used. This language also allows better integration with the iPhone OS API than C or C++.

The NX Framework's architecture is presented in figure 2. It comprises the Engine and the Network modules. The former contains classes and protocols common to 2D game development while the latter specificaly emcompasses the network infrastructure of the framework. This architecture's proposal offers a compact set of abstractations of the Apple's frameworks (present in the lowest-level layer) that allows the quickly development of new games, by just adopting the NX Framework protocols.

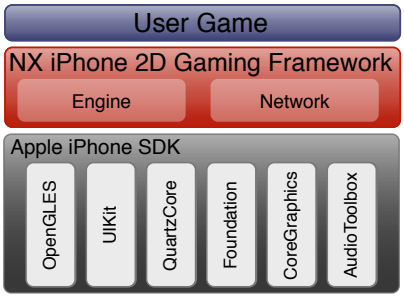


Figure 2: *NX Framework architecture*

4.1 Engine module

The **Engine** module is the game engine properly speaking, it is responsible for controlling the global functionality of a 2D game and provides classes for the game instantiation, user multi-touch event handling and game screen management. Figure 3 shows the class structure for the Engine module.

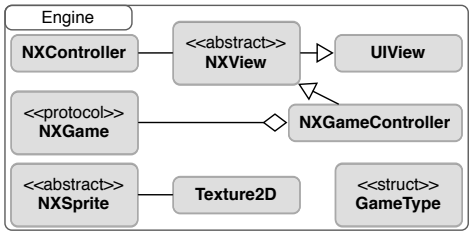


Figure 3: *Engine module class structure*

The **NXController** is the module's main class, it controls the game displaying on iPhone screen and maps the events that arises in the visualization layers to the game implementation. It also makes the transitions between classes that inherit from NXView, mananging their allocated memory. Moreover, the **NXController** class has operations that allows the presentation of game screens that can be loaded from a NIB file. Only one object of this class is instantiated and its life cycle corresponds to the whole application life cycle. Since this object has a reference to a NXNetwork class instance, it can provide to the other game classes the network communication infrastructure, if needed. At last, the **NXController** class has data persistence mechanisms, which allow game state persistence that would be recovered in a future execution.

The abstract class **NXView** represents a game view that can be displayed on the iPhone screen. It inherits all functionalities from UIView class (UIKit framework) and has, in addition, a reference to a NXController object. As a result, all its descendent classes are able to use their reference to the NXController object to request a transition between NXViews, which occurs, for instance, when the screen is changed from the game screen to the options menu screen. Interface files constructed in Apple's Interface Builder (NIB files) typically describes GUI screens. They can be loaded and rendered with the creation of NXViews sub-classes which are the NIB's "file owners".

NXGame is a protocol[2] that defines the operations that must be implemented by a 2D game. These operations comprise memory management issues and, mainly, the gameloop. This protocol also garantees that classes that adopt it have a reference to a NXGameController object, in this way, these classes are able to suspend or pause the gameloop.

The **NXGameController** is a NXView specialization for a CAEAGLLayer layer that creates an OpenGL ES context. In other words, it is a view that displays OpenGL ES content on the iPhone screen. Its main function is to instantiate the game and manage the gameloop by using control and state attributes. The actual game to be controlled is an instance of a class that adopts the NXGame protocol. Since this class specification is generic, its constructor requires a `GameType` data structure in order to know what kind of game to instantiate. The `GameType` data structure holds the game's class and whether or not it requires network communication. Finally, the abstract class **NXSprite** defines the minimal characteristics for a 2D game object, and so aggregates a Texture2D object, which is responsible for text and texture drawing. In addition, the **NXSprite** class holds the basic attributes that represents a 2D game entity. The simulation and rendering operations are defined as abstract and can be overwritten if needed.

4.2 Network module

This module provides a basic structure to create multiplayer games using client-server networking communication. It was developed using iPhone SDK classes `NSNetService` and `NSNetServiceBrowser` which adopt the zero-configurantion standard. Also, this module defines a specific communication protocol for game data exchange.

4.2.1 Zero-Configuration

Zero-configuration or Zeroconf is a network communication architecture. It's an IETF standard to manage TCP/IP networks without needed of manually configuration or a network administrator. Its goal is to let users connect their computers or devices in a local network – by Ethernet or Wireless connection – and gain access to use all available local network services. To achieve the current Zeroconf pattern, the operational system or device have to implement three functionalities: *i)* be capable to self-assign an IP address without a DHCP server (addressing); *ii)* Translate names to IP addresses without a DNS server (naming) and *iii)* discover available local network services (service discovery). Promoted by Apple Computer Inc., it is available as the Bonjour implementation [Voip-Info 2009].

4.2.2 Bonjour

Bonjour implements Zeroconf's functionalities of addressing, naming and service discovery. To addressing, the Bonjour's proposed solution is the auto-assign of IP addresses into a LAN or a network segment. The naming process is similar to the addressing one – each service or device auto-assigns a name and tests if it is already in use. Finally, the service discovery lets applications search into local network for a particular service type's instances and keep on a list of services names (which are persistent rather than non-persistent addresses), allowing a service name to be resolved in an

[2] A protocol, in Objective-C, is a list of method declarations that any class and perhaps many classes, might implement. A protocol is simply a list of method declarations, unattached to a class definition.

address and a port number always that is needed [Apple Computer 2009b].

Differing from the traditional devices oriented approach, Bonjour is service oriented. This makes requests to all devices about "what services they provide" unnecessary, just requesting about "which device provides determined service". To set a service instance name, Bonjour uses the convention: "`ServiceName._ServiceType._TransportProtocol-Name.Domain`". A valid example could be "`I-601's iPhone._airhockeygame._tcp.local.`", which represents an `airhockeygame` service type available by `TCP` connection in the `local.` domain, where the `ServiceName` is a human readable descriptive name. Each element is separated by the "_" character. Bonjour services architecture supports three basic operations: service publication, discovery and resolving.

4.2.3 Protocols

This network module of NX Framework is composed by classes presented in figure 4 and the communication protocols described below:

- NXServerDelegate: this protocol defines the operations `serverDidEnableBonjour:withName:`, `server:didNotEnableBonjour:` and `didAcceptConnectionForServer:inputStream:outputStream:` which must be implemented in a delegate class to manage a server creation and a Bonjour service publication. These operations are invoked if a Bonjour service was published, was not published or when it receives a connection, respectively;

- NXClientDelegate: defines the operations `netService:DidResolveAddress:` and `netService:DidNotResolve:` which must be implemented in a delegate class to manage a Bonjour service resolution. These operations are invoked when a Bonjour service was or was not resolved, respectively;

- NXNetworkController: defines the operation `setupNXNetwork:` which must be implemented in a class that has to control the NXNetwork object. Through this operation, the controlled object must be initialized and informed that a class which adopt this protocol is its current controller;

- NXNetworkServerController: is a NXNetwork-Controller's specialization, defining the operations `serviceDidCreate:`, `serviceDidNotCreate:` and `serviceDidAcceptConnection:` which must be implemented to control the Bonjour service publication process. These described operations are invoked by the controlled object, respectively, when a Bonjour service was or wasn't published or when a connection to a published service was established, allowing the decision making for the class that adopt this protocol;

- NXNetworkClientController: is a NXNetwork-Controller's specialization, defining the operations `serviceDidResolve:`, `serviceDidNotResolve:`, `didConnectToService:` and `didNotConnectToService:`, which must be implemented to control the Bonjour service resolution and connection processes. These operations are invoked by the controlled object, respectively, when a Bonjour service was resolved as well as when a connection to a Bonjour service previously resolved was established;

- NXStreamEventHandler: define the operation `stream:handleEvent:` that controls the stream of bytes through the input and output streams available in a NXNetwork instance. These streams are created after a connection to some Bonjour service's server.

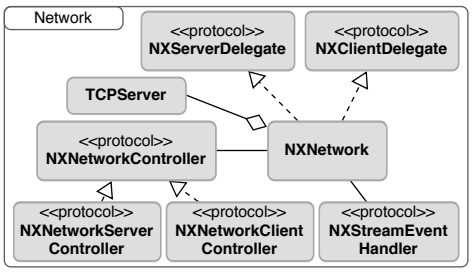


Figure 4: *Compact Network module's class diagram*

4.2.4 The NXNetwork class

Having commonly only one instance in an application, this class adopts the `NXServerDelegate` and `NXClientDelegate` protocols, allowing itself to publish, discover or resolve a Bonjour service. Once defined the domain, the service type and the transport protocol, it becomes possible to execute the three basic Bonjour operations (service publication, discovery and resolution) through the operations `publishService:` and `browseAndResolveService:`. The operations are performed as described bellow:

- publication: instantiates a TCPServer class object – responsible to create a TCP server – and publics the Bonjour service by the `publish:` operation from the `netService` attribute;

- service discovery: uses an instance of `NSNetServiceBrowser` – responsible to return all Bonjour services of a selected type – to update the `netService` attribute with the desired service;

- resolution: calls the `resolveWithTimeout:` operation from the `netService` attribute.

During the execution of the above operations, the reported events are notified to the `activeNetworkController` attribute – a reference to an object that adopts the `NXNetworkController` protocol – which should be informed through the `setActiveNetworkController:` operation. After a connection through the `connectToService:` is established, this class is responsible for opening the input and output streams – instances of `NSInputStream` and `NSOutputStream` – and resend these streams' reported events to the `activeStreamEventHandler` attribute. This attribute keeps a reference to an object that adopts the `NXStreamEventHandler` protocol, which should be informed through the `setActiveStreamEventHandler:` operation to transmit bytes over a network. Lastly, the `stop:` operation finishes the current active service.

5 Samples

At the moment two game projects were developed using the NX Framework. The first one, Zig Zig Zaa [Coelho et al. 2009] is an application that was designed for educational purpose and contains two embedded games. It is already published on Apple's App Store. The second one is an AirHockey game (named here as NX-AirHockey), which will be introduced here as a study case.

In the NXAirHockey game, matches can played by two players simultaneously in the same device (by using the multi-touch capabilities) or even on different devices (by using the NXNetwork infrastructure). A single player game mode is also supported, in this case, the player plays against the computer that is controlled by the artificial intelligence module that was implemented. The game architecture comprises two modules: Model and Game. The Model module aggregates the game domain entities, which are subclasses of NXSprite. The Game module actually do the game implementation by making the interaction among the entities defined in the Model module. It also encompasses the game menus and connection screens, along with the gameplay screen itself. The class diagram shown in figure 5 depicts the classes responsible by the game-

play implementation. The `NXController` class has the function of present to the user the game graphical content on the iPhone's screen. In this case, this class aggregates a `NXGameController` object, which is a specialization of a `NXView` for a CAEAGLLayer layer. This is due the fact the game rendering is done by OpenGL ES.

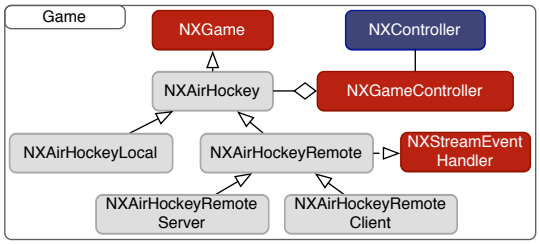


Figure 5: *NXAirHockey Game module (gameplay)*

The `NXAirHockey` class serves as the basis for the other game implementation classes. It adopts the `NXGame` protocol and therefore implements the gameloop and event handling operations. By doing this, it is capable to manage the game rules. Some specializations of this class were made in order to distinguish instances of games that occur locally or remotely.

The class `NXAirHockeyLocal` comprises the implementation of a game that occurs locally, either in single player or multiplayer mode. Regarding the implementation of a remotely game, after established a client-server connection between two game instances running on different devices, confirmation messages are sent through the input and output opened streams, in order to know the moment the game can start. Afterwards, a `NXAirHockeyRemote` descendant class object is instantiated, a `NXAirHockeyRemoteServer` instance in the case of a game server instance or a `NXAirHockeyRemoteClient` in the case of a client game instance.

The `NXController` class is capable to present OpenGL based views as well as views composed by UIKit components (NIB files) into the iPhone's screen. Thus, menus and screens similar to a connection screen were generated in the Interface Builder, hence being easily and quickly developed and edited. This approach is particularly desirable because UIKit components are more functional for user interface such as menus and configurations screens than Open GL. The figure 7 presents the `NXController` showing the two different screen types. View transitions are smoothed through Core Animation (QuartzCore framework) features.

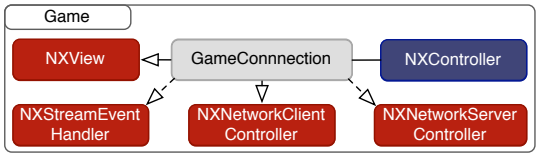


Figure 6: *NXAirHockey's game module*

The figure 6's diagram shows the `GameConnection` class being presented into iPhone's screen by the `NXController` (figure 7) and also presents the class managing connection between different game instances, through the protocols `NXNetworkServerController`, `NXNetworkClientController` and `NXStreamEventHandler`. In addition, the class is responsible for publishing or resolving a Bonjour service, respectively, when running in server or client mode.

6 Conclusions

This paper discussed the development of the NX 2D Gaming Framework, which brings an alternative option for the development of smaller scale game projects. Comparing to the development based purely in iPhone's SDK APIs, our framework offers some advantages such as a network communication abstracion layer based

Figure 7: *(a) Game connection screen (loaded from a NIB file); (b) NXAirHockey game screen (rendered into an OpenGL canvas)*

on Zeroconf standard, integration with user interfaces constructed with Apple's Interface Builder and integration with Open GL ES. The use of NIB files (from Interface Builder) for non gameplay classes leads to a faster development of menu and others views. This advantage was not found in the related works. Future improvements include game interface support based on iPhone's accelerometers and support for audio and music, which is currently made by third party libraries.

Acknowledgements

The authors would like to thank University of Blumenau (FURB) and Project Acredito for the finantial support for equipment aquisition.

References

ALLEN, C., AND APPELCLINE, S. 2008. *iPhone in Action: Introduction to Web and SDK Development*. Manning Publications Co., Greenwich, CT, USA.

APPLE COMPUTER, 2009a. iPhone OS technologies. `http://developer.apple.com`, May.

APPLE COMPUTER, 2009b. Bonjour overview. `http://developer.apple.com`, April.

COCOS2D, 2009. cocos2d for iphone. `http://www.cocos2d-iphone.org/`, June.

COELHO, E., PIVA, G. R., AND GOMES, P. C. R., 2009. ZigZigZaa - Apple App Store. `http://itunes.apple.com/WebObjects/MZStore.woa/wa/viewSoftware?id=306502103&mt=8`, Mar.

DALRYMPLE, M., AND KNASTER, S. 2008. *Learn ObjectiveC on the Mac*. Apress, Berkely, CA, USA.

ENGEL, W., 2009. Oolong engine. `http://oolongengine.com/`, June.

GARAGEGAMES, 2009. iTGB. `http://www.garagegames.com/products/torque-2d/iphone`, June.

SIO2, 2009. Home. `http://sio2interactive.com/`, June.

STONETRIP, 2009. Stonetrip: Development is a game. `http://www.stonetrip.com/`, June.

UNITY TECHNOLOGIES, 2009. Unity: Game development tool. `http://unity3d.com/unity/`, June.

VOIP-INFO, 2009. Asterisk zeroconf support - voip-info.org. `http://www.voip-info.org/wiki/view/Asterisk+Zeroconf+Support`, June.

Towards Virtual Actors - The Next Step for the Entertainment Industry

Rogério E. da Silva
University of Minho
Guimarães, Portugal
Santa Catarina State University
Joinville, Brazil

Ido A. Iurgel
University of Minho
Guimarães, Portugal

Manuel F. dos Santos
University of Minho
Guimarães, Portugal

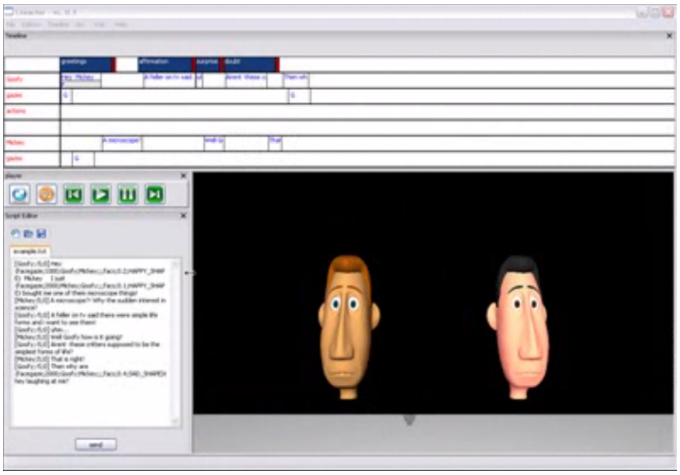


Figure 1: *CREACTOR Animation Tool interface*

Abstract

Autonomous Digital Actors (ADA) represent the next step for the entertainment industry in the sense that they are new kind of virtual characters endowed with self-animation capabilities delivered by artificial intelligence techniques. There are many substantial open questions related to the development of virtual actors. This article introduces the subject 'virtual actors' and describes an experimental authoring tool, called CREACTOR. We are developing this tool for studying which AI techniques can be used to design and implement behaviors of ADAs.

Keywords:: Interactive Storytelling, Computer Animation, Believable Characters, Virtual Actors

Author's Contact:

rsilva@joinville.udesc.br, idoiurgel@yahoo.de, mfs@dsi.uminho.pt

1 Introduction

In [Perlin and Seidman 2008], the authors foresee that "*3D animation and gaming industry will soon be shifting to a new way to create and animate 3D characters, and that rather than being required to animate a character separately for each motion sequence, animators will be able to interact with software authoring tools that will let them train an Autonomous Digital Actor (ADA) how to employ various styles of movement, body language, techniques for conveying specific emotions, best acting choices, and other general performance skills.*"

Our ongoing research aims at developing virtual actors. Virtual actors are a kind of believable characters [Reilly 1996]. A virtual actor is an analogy to a real actor, which autonomously, and by its independent interpretation of the situation, can perform its role according to a given script, as part of a story [Iurgel and Marcos 2007; Iurgel 2008].

Virtual actors must exhibit a behavior similar to real actors, which means, they must be able to play roles specified according to guidelines; but also they must be able to use their own previous experiences, adapting their performance to new contexts. Besides that, they should also be capable of understanding abstract commands from a director/animator who could inform them about desired behaviors in a particular situation.

The ability of characters to improvise and get guidelines from a director represents a new approach on authoring characters for the production of 3D-movies. Through this new approach, it should be possible to speed up the process of creation of animation films populated with synthetic characters, because those characters are able to decide about most of the acting performance by themselves, reducing the time and effort needed for animation.

We are developing an authoring framework, called CREACTOR (see figure 1), that is an experimentation platform for exploring different solutions and processes. The actors are implemented as talking heads. CREACTOR focuses on providing a broad number of features and modules that are at first implemented in a shallow way. Future steps shall then identify the most important dependencies and possible enhanced solutions on the level of the modules.

The expectation is that this kind of tool will simplify the authoring process in such a way that unskilled animators will be able to create 3D animations with a higher quality than current animation tools allow.

2 Virtual Actors

A real actor is someone capable of expressing dramatic actions (theatrical skills) based on a script. A virtual actor is a believable character capable of understading its role from an annotated script and, by interpreting the situation, to propose an appropriate behavior (acting performance) for it. This performance could then be corrected/improved by the animator.

Next sections aim at explaining our actors/humans metaphor, how acting students learn how to act (this process had inspired us to propose our method for creating virtual actors), and then how to implement virtual actors.

2.1 (Virtual) Actors vs. (Virtual) Humans

An actor has abilities that a normal person does not have, for instance, an actor can play a role.

Virtual humans are, according to Thalmann & Thalmann [Thal-

mann and Thalmann 1993], "a visualization of the simulation of the behavior of realistic human beings". So, in other words, virtual humans try to replicate as accurate as possible real persons. There are three levels of modeling concerning the development of virtual humans [Thalmann and Thalmann 2005]: (1) realistic appearance modeling, (2) realistic, smooth and flexible motion modeling and (3) realistic high-level behaviors modeling.

Virtual actors, on the other hand, are a different (simpler) approach towards the creation of embodied autonomous characters capable of performing specific roles in a story. They are, for that matter, a specialization of virtual humans. A virtual actor should be capable of:

- performing a given role based on a knowledge base that describes it (e.g. playing a surgeon);
- understanding a script that contains all the utterance and actions that must be played during a scene of a film (e.g. performing a brain surgery);
- interpreting the context of a scene being played and, by doing so, deciding (suggesting) the most appropriate behavior to perform (e.g. displaying an anxious face due to complications during the procedure);
- getting suggestions (from the director) that could modify/improve its behavior during the performance (e.g. playing the scene as a bad temper doctor).

2.2 Acting Schools

There are basically two different acting schools [Roberts 2007]:

Method Acting: it is a method proposed by Konstantin Stanislavsky that states that to develop a role, an actor should "use his or her own emotional experience and memory in preparing to live a role". This approach develop roles from the inside out and means to decide 'what are the characters' motivations for the scene?'.

Theatrical Acting: as opposite to method acting, this approach claims that developing a role requires learning a series of positions and facial expressions inferred from the script. This approach relies on an answer to the question 'what are the appropriate behaviors for this scene?'.

Method acting seems to point to a more 'accurate' acting performance since the actor spontaneously reflects his/her body postures, facial expressions and voice intonation according to the emotions being felt. As a consequence new situations should not mean having to learn new behaviors.

In theatrical acting, an actor must know everything about the role being played in order to be able to act believably. That means that new situations require new information about appropriate responses.

2.3 Virtual Actors Requirements

Loyall [Loyall 1997] proposes a set of requirements in order to achieve believable characters, that certainly should also be applied to virtual actors. The actual set of requirements needed for the creation of a virtual actor will depend on the acting approach adopted. This is the list of requirement of Loyall:

- **From the artistic point of view:**

  Personality: Loyall cites Thomas and Johnston (two important Disney animators), who said that being able to represent different actions for different characters in the same situation is one of the most important aspects of promoting the illusion of life.

  Emotions: every character should express (and maybe has) emotions based on its own personality.

  Self-motivation: the character should act as if it comprehends what and why it is doing.

  Change: To be believable, a character should grow with time or life experiences. However, this growth should be related to the character's personality and to the story.

  Social relationships: characters must comprehend how to interact with other characters and treat each one differently.

  Consistency of expression: To promote believable actions, all communication channels (facial expressions, body posture, etc.) should be simultaneously connected to each other to produce a coherent result.

- **From the autonomous agents' point of view:**

  Appearance of goals: every agent should have goals (purposes) that will motivate its actions and decisions.

  Concurrent pursuit of goals and parallel actions: every living creature is able to perform several different actions simultaneously.

  Reactive and responsive: every time an agent perceives an event it should react to it. Reactive behavior is natural and quite common today in classical digital entertainment applications. However, it is not enough. An agent should respond to events based on some sort of cognitive consideration.

  Situated: agents should be prepared to change what and how they are doing it in response to the unfolding of situations.

  Resource bounded (body and mind): every person has physical and mental limits.

  Existence in a social context: a character should know and respect social conventions, laws, rules, cultures, etc. of the world it is immersed into.

  Broadly capable: characters should be flexible enough to support a significantly large set of actions.

  Well integrated: its actions should be performed in a smooth, realistic manner.

3 Preliminary Results

The project is devising an authoring framework for interactive, adaptable, situation aware, partly autonomous virtual actors (cf. also [Iurgel and Marcos 2007]). The animator/director shall direct virtual actors by providing them with the text to speak, explaining the dramatic situation (using concepts of an ontology and GUIs for this); the actors will try to infer the appropriate expressions, animations, and the timing (our work is related to [Perlin 2003] and [Mateas and Stern 2003], though Perlin focuses on scripting methods and Mateas on maintaining a plot structure). CREACTOR, the authoring framework, is an experimentation platform for exploring different solutions and processes. It focuses on providing a broad number of features and modules that are at first implemented in a shallow way. Future steps shall then identify the most important dependencies and possible enhanced solutions on the level of the modules.

The authoring process is a cycle in which three professionals are involved (refer to figure 2):

Script Writer: is the person responsible for writing the script. It is a limitation of this work that it only considers conversational behavior.

Acting Teacher: is the person responsible for creating all characters' acting skills, in other words, creating the knowledge that allows virtual actors to know how to behave in any given situation during the performance, for a specific domain.

Animator/director: is the person responsible for adjusting the actors' performance according to the desired outcome.

All professionals could/should interact with each other, making this process iterative (as represented by the dotted lines). The goal is

that the number of cycles should be small enough that to justify its use in place of the 'traditional process'.

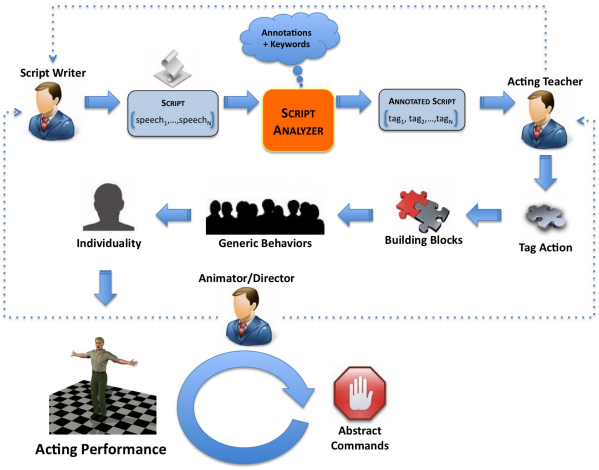


Figure 2: *Virtual Actors Authoring Process*

The authoring process can be divided into three major steps: script annotation, character specification and acting performance.

3.1 Step #1: Script Annotation

First, script writer writes a script of the scene, as a series of speech sentences, represented in figure 2 by the tuple $[speech_1, \ldots, speech_N]$. These sentences should then be performed by the actors during step #3. However, to be able to accomplish that, they should be capable of analyzing and understanding the meaning of each sentence.

Specifically regarding analyzing sentences, the system will try to infer an adequate annotation (semantic/pragmatic) for each sentence. It is not an easy task,and we are approaching the problem through very simple heuristics. We were inspired by the A.L.I.C.E. bot[1], and its Artificial Intelligence Markup Language (AIML).

We employ a simplified markup language that tries to match keywords to sentences. The AIML tags were renamed as follows:

config: enclose an entire document

annotation: marks an unity of annotation, i.e., one possible type of action performable by the character.

keyword: a sentence (keywords) that, if matched, indicates that this annotation should be used

The following incomplete code exemplifies the annotation language in use. Each keyword is evaluated considering a non case-sensitive criteria using whole words only; however, it is possible to use the wildcard "*" within each keyword to overcome this issue.

```
<config>
  <annotation name="admire">
    <keyword>!</keyword>
    <keyword>wow</keyword>
    <keyword>amazing</keyword>
    <keyword>incredible</keyword
    <keyword>awesome</keyword>
    <keyword>cool</keyword>
  </annotation>

  ...
</config>
```

Each speech sentence will enclose an 'annotation DNA', that represents how often the sentence is related to each possible annotation, creating a semantic map of the given annotation. For example, let us say that the sentence 'Hey Mickey' matched the annotations `greeting` and `complaint` and no other. The system will then understand that this sentence means one of these two meanings and has to decide for one of them.

In our case, what is being used for choosing between possible meanings is a weightened random choice. This means that the system will choose randomly but will take the amount of times the sentence were tagged with a particular annotation as a weight to increase the chances of choosing it. So, if a sentence is tagged twice as greeting and just once as complaint, there are 2/3 odds that the system will consider it as a greeting.

Another way of (theoretically) increasing the confidence in the computer's choice is to associate a degree of confidence for each pair `<annotation, keyword>`. This degree would represent how sure we are that the keyword relates to the annotation. So, this degree could then be used instead of the occurrences counter approach described before.

3.2 Step #2: ADA Specification

This step is responsible for defining 'how' it should do it. In other words, how a character behaves when it (for example) is insulting or mocking another character.

It is important to notice that the set of rules that compose the repertoire of behaviors that a character has, is what enables it to play a role in a given story.

So, these rules must be designed in such a way that they should be modulated by the character's individuality (personality traits and inner emotional state).

3.3 Step #3: Directing the Performances

CREACTOR shall enable the author to define the performance at various levels of abstraction. These definitions range from abstract commands from a director to specific changes at the level of the body of the virtual actors. The authoring process shall be iterative, because of the necessity of constant visual feedback of the creative work; assisted, because of the manifold alternative steps that the author can take at every moment; and, in a later step, dialogic, in the sense that the system shall be able to urge the author to provide additional information about his/her intentions, in order to enable the system to complete underspecified animations autonomously.

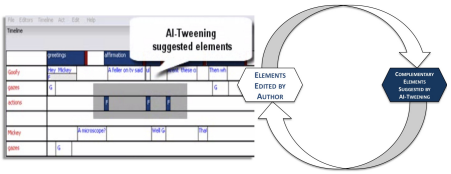


Figure 3: *AI-Tweening Concept*

As an example of our current work, we present here the concept of "AI-tweening", that serves also to illustrate the intended authoring process: CREACTOR assists the author by indicating that the user first needs to define the main plot points, and then to specify the desired behavior of the virtual actors for these plot-points. The system then produces an estimation, based on AI-knowledge, of intermediate behaviors. This is the process of "AI-tweening". For instance, between an angry and a calm behavior, the system inserts a "cool down"-transition. The autonomous generation of "AI-tweens" serves as basis for the author to further enhance the animation. In a future step, the system shall be able to actively gather additional required information, asking for instance whether there is a turning point that causes the anger to disappear, or whether it is a gradual fading away.

3.3.1 Coherence Analyzers

There are two types of analysis under AI supervision: *vertical coherence* means that an actor should maintain coherence between high and low level commands, after changes of a single command

[1] `http://alicebot.blogspot.com/`

by the user. *Horizontal coherence* means that the transitions between behaviors (over time) will 'make sense' based on a specific set of rules that denote how a virtual actor should behave.

As an example, let us consider that the user changes a command from happy to sad. Hence, the character's face will autonomously correct its setup, and possibly the autonomous interpretation of the situation, to match the new input.

Horizontal coherence works similarly on character's behaviors over time. Thus, from a plot that expresses anger at the beginning of the film to another one that expresses calmness at the end, the system will insert plots that would represent a "cool-down" transition. Analogously to vertical coherence, any changes would cause autonomous adjustments in between the plots.

4 Conclusions

The entertainment industry constantly evolves aiming at creating more appealing believable characters. Autonomous digital actors (ADA), that according to [Perlin and Seidman 2008; Iurgel and Marcos 2007], represent a new way to create and animate 3D characters, should be capable of interacting with animators and displaying behaviors that will significantly simplify the authoring process of character-based animations.

This article has presented an overview regarding these autonomous digital actors (here called simply virtual actors). It is defined virtual actors, explained the metaphor recurring to real actors/humans, listed the major requirements (according to the literature).

Finally, an ongoing research on the design and implementation of virtual actors was sketched, and we have discussed an authoring process that consists of three steps. A preliminary tool called CREACTOR was implemented. It aims at serving as exploratory environment for testing hypothesis raised during the research. However, at this point, the system only considers manually annotated scripts and rule-based AI-tweening. The current AI-tweening allows only for limited behavior suggestions.

The next steps include the implementation of the autonomous textmining-based annotation module and the knowledge database and reasoning rules.

Acknowledgements

This work was partially funded by the Portuguese Science and Technology Foundation (FCT) under the reference PTDC/EIA/69236/2006.

References

IURGEL, I. A., AND MARCOS, A. F. 2007. Employing personality-rich virtual persons—New tools required. *Computers & Graphics 31*, 6 (Dec.), 827–836.

IURGEL, I. 2008. VirtualActor: endowing virtual characters with a repertoire for acting. In *Interactive Storytelling*, Springer Verlag, Erfurt, Germany, U. Spierling and N. Szilas, Eds., vol. LNCS 5334, 89–91.

LOYALL, A. B. 1997. *Believable Agents: Building Interactive Personalities*. PhD thesis, Carnegie Mellon University. 222 pages.

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

PERLIN, K., AND SEIDMAN, G. 2008. Autonomous digital actors. In *Motion in Games*. 246–255.

PERLIN, K. 2003. Building virtual actors who can really act. *Virtual Storytelling*, 127–134.

REILLY, W. S. N. 1996. *Believable Social and Emotional Agents*. PhD thesis, Carnegie Mellon University.

ROBERTS, S. 2007. *Character Animation: 2D Skills for Better 3D*, second ed. Focal Press.

THALMANN, N. M., AND THALMANN, D. 1993. The world of virtual actors. *Virtual Worlds and Multimedia, John Wiley*, 113–126.

THALMANN, N. M., AND THALMANN, D. 2005. Virtual humans: thirty years of research, what next? *The Visual Computer 21*, 12, 997–1015.

Um Jogo de Caça ao Tesouro Utilizando o Google Earth

Antônio C. L. Araújo[1], Helder G. Aragão[1,2], Marcos L. dos Santos[1],
Roberto C. S. de Jesus, Tácio F. Silva[1]

[1]Faculdade de Ciências Exatas e Tecnologia (FCET)
Centro Universitário da Bahia (FIB)
Salvador - BA

[2]Grupo de Aplicações e Análise Geoespaciais (GANGES)
Universidade Salvador - Unifacs
Salvador - BA

Abstract

Caça Tesouro is a game of general knowledge that is played at Google Earth - a Geographical Information System (GIS) for desktop environment, controlled by an Web application. After been registered the game it starts by the user through the website and, the search for the "Treasury" will be for the clues that are given to them at Google Earth's screen. When the Player finds the specified place he receives a new clue that contains an indication for the next place to be located. The game finishes when the user finds the hidden "treasure" acted by the last clue.

Keywords: Google Earth, SIG, KML, game.

Resumo

Caça ao Tesouro é um jogo de conhecimentos gerais que é jogado no console do *Google Earth* - um Sistema de Informações Geográficas (SIG) para ambiente *desktop,* controlado por uma aplicação Web. Este jogo, após um cadastro do usuário, é iniciado através do *website* e, a busca pelo "Tesouro" se dará pelas dicas que lhes são fornecidas na tela do *Google Earth.* Quando o Jogador encontra o local especificado, ele recebe uma nova dica que contém uma indicação para o próximo lugar a ser localizado. O jogo termina quando o usuário encontra o "tesouro" escondido representado pela última dica. O jogo possui em sua arquitetura um Web Service que valida se o Jogador encontrou os lugares corretos.

Palavras Chave: Google Earth, SIG, KML, Jogo Educativo.

Contatos dos Autores:
helderaragao@yahoo.com.br
{marcoslapa,aclaraujo,robertocarlossj}
@gmail.com
toko@ig.com.br

1. Introdução

O georreferenciamento está cada vez mais em evidência, motivando o interessante no desenvolvimento de aplicações que possam se beneficiar do mesmo. A Google Inc. por ser uma empresa que está crescendo a cada dia, popularizou muito esse novo conceito de aplicações ao disponibilizar, gratuitamente, as ferramentas de navegação com mapas como *Google Maps*, e logo depois o *Google Earth* [Google 2008]. Estas duas aplicações podem ser classificadas como Sistemas de Informações Geográficas (SIG), por fornecerem aos usuários ferramentas de manipulação de dados geoespaciais [Pina e Santos 2000]. Existem outras aplicações nesta mesma linha disponíveis, tais como a *Microsoft Virtual Earth* e o *ArcGIS Explorer*.

O objetivo do presente artigo é demonstrar a implementação de um jogo, chamado Caça Tesouro, para Google Earth. Será mostrada a interoperabilidade deste jogo com o Google Earth e a sua característica lúdica adequado ao contexto

educacional. O artigo está dividido desta forma: a seção 2 aborda conceitos relativos aos *virtual globes*; a seção 3 descreve o Google Earth; a seção 4 mostra o uso e a implementação do jogo e, por fim, na última seção estão descritas as conclusões e os trabalhos futuros.

2. Virtual Globes

Virtual Globes são programas que proporcionam a visualização do globo terrestre através de imagens bidimensionais (2D) e tridimensionais (3D) [Virtual Globe 2009]. Estes programas se enquadram no grupo dos Sistemas de Informações Geográficas (SIGs) por fornecerem algumas funcionalidades típicas dos SIGs, tais como *zoom-in*, *zoom-out* e *pan*. Apesar de já existirem a algum tempo, a popularização dos *virtual globes* só aconteceu após a disponibilização de aplicações gratuitas, tais como *Google Earth, Google Maps, Microsoft Virtual Earth,* entre outros. Isto atraiu novos usuários, que atualmente conseguem interagir e manipular informações espaciais sem grandes dificuldades devido a uma baixa complexidade desses aplicativos.

Os *Virtual Globes* oferecem recursos, além da visualização do globo, de localização, com a qual é possível encontrar, por exemplo, cidades, estradas, hospitais, escolas, restaurantes e shoppings.

3. Google Earth (GE)

O *Google Earth* é um *virtual globe* para ambiente *desktop* desenvolvido inicialmente pela Keyhole Inc. que o batizou como o nome de *Earth Viewer*. A *Keyhole* foi absorvida pela Google em 2005 que o rebatizou com o seu nome atual [Silverman 2008; Brown 2006].

O *GE* funciona de forma semelhante ao *Google Maps* oferecendo mais recursos, dentre os quais, destacam-se o Passeio Virtual e Simulador de Vôo. O Passeio Virtual permite ao usuário sobrevoar por uma rota criada a partir de um ponto inicial e final. O *Simulador de Vôo* dá ao usuário a sensação de estar pilotando um avião sobre o globo [Google 2008].

Para a visualização dos dados geográficos no GE, deve-se utilizar um arquivo padrão em formato XML (*eXtended Markup Language*) chamado de KML (*Keyhole Markup Language*). Este arquivo foi criado originalmente para o GE e foi especificado como um padrão pela *Open Geospatial Consortium* (OGC), consórcio responsável pela padronização dos formatos dos dados geográficos [OGC 2007; Erle e Gibson 2006].

A escrita KML é bem semelhante a do XML e apresenta como principal diferença o fato de suas marcações serem pré-definidas para um fim específico (aplicações georreferenciadas). Uma destas marcações é a <kml>, que distingue um documento KML de um XML comum. A escrita KML também se assemelha ao *HyperText Markup Language* (HTML), linguagem de marcação utilizada na Internet [Erle e Gibson 2006].

O KML pode ser considerado um formato leve, de fácil manipulação e envio pela rede. Este formato conta também com validações baseadas em *XML Schema* (que é uma espécie de verificador sintático como os das linguagens de programação), executadas pelo próprio servidor do *Google Earth* através do *namespace* especificado dentro do próprio arquivo KML. Isto permite detectar erros de sintaxe nas marcações XML [Rehem 2008]. Mais detalhes sobre esta especificação podem ser encontrados no *KML 2.2 – An OGC Best Practice*, o guia de melhores práticas da OGC [OGC 2007]

4. O Jogo Caça Tesouro

Caça Tesouro é um jogo desenvolvido para Web utilizando o *framework* .NET, o qual interage com o *Google Earth (GE)*. Essa interação acontece através de arquivos KML gerados pela aplicação e executados no GE. A linguagem de programação

escolhida para este projeto foi o *C#* [Microsoft 2009]. Com esta linguagem foi desenvolvido um componente chamado *KML Generator,* que tem a função de gerar os arquivos no formato KML para serem interpretados pelo GE.

4.1 O Objetivo do Jogo

O Jogo Caça Tesouro tem por objetivo encontrar o "tesouro" escondido em um determinado lugar do globo terrestre no menor tempo possível. Essa busca é feita no próprio console do *Google Earth* através das dicas que são fornecidas pelo controlador do jogo, uma aplicação Web, que permite essa interação.

4.2 A Jogabilidade

O Jogador deve localizar os lugares previamente determinados espalhados pelo globo terrestre através de dicas fornecidas pela aplicação. Quando o jogo se inicia, o Jogador recebe a primeira dica, com informações para sua busca inicial, descrita através de um marcador de lugar (placemark) do *Google Earth* (GE).

O Jogador deve navegar pelo globo terrestre em busca do local indicado na primeira dica e ao encontrá-lo, o jogo enviará a segunda dica, a qual surgirá na posição exata indicada anteriormente. As próximas dicas são exibidas à medida que o usuário localiza os lugares indicados.

Para tanto, o Jogador deve utilizar a função de *zoom-in* disponível no GE, com a qual é possível se aproximar do globo terrestre. Isso porque, para que as dicas sejam exibidas, a imagem deve ser mostrada com a câmera do GE posicionada a certa distância do solo, já determinada pelo grau de dificuldade escolhido no jogo, de modo que a diferença entre os pontos oeste e leste da imagem atual, exibida no GE e dada em quilômetros, seja menor que a distância máxima estabelecida nas regras do jogo. Portanto, não é possível receber as próximas dicas sem que se procure os lugares, minuciosamente, sugeridos por cada dica. Por exemplo, se a distância máxima para a visualização é equivalente a 2 km, a imagem mostrada deve ser aproximada de forma que exiba aproximadamente 2 km² de área (Figura 1).

Figura 1: Terceira dica do jogo exibida sobre a Estátua da Liberdade.

A quantidade total de lugares a serem descobertos pode variar, dependendo do grau de dificuldade. Quando o Jogador encontrar todas as dicas, o jogo termina e a última dica confirma que o "Tesouro" foi encontrado.

Caso deseje jogar novamente, o Jogador deve selecionar a opção de um novo jogo. Ao iniciar um novo jogo, novas dicas serão sorteadas para que o Jogador possa jogar mais vezes com um menor número de repetições, até que ele esgote todas as missões cadastradas. As dicas sorteadas se encontram armazenadas em um banco de dados relacional acessado através da aplicação controladora do jogo.

O Jogador conta com um visualizador de histórico no site do jogo para relembrar as dicas encontradas e a última a encontrar. O Jogador pode, também, verificar este histórico no próprio GE no painel *Lugares*.

4.3 A Arquitetura

A Figura 2 mostra uma visão geral da arquitetura do jogo Caça Tesouro. O jogo, representado pelo navegador Web (item 01), utiliza o *KML Generator* (item 02), que é um componente customizado com a função de gerar os arquivos KML necessários para exibição das mensagens na tela do Google Earth. Essa comunicação é realizada através de um *Web Service* (item 03) criado especificamente para fazer uma interface entre a aplicação Web controladora do jogo e o GE (item 04). Ao localizar uma dica no globo, o GE se comunica com o *Web Service* que fará a validação do lugar encontrado. Caso o ponto a ser encontrado esteja sendo visualizado no globo, a uma distância menor que a máxima determinada, o ponto é localizado e o Jogador receberá a próxima dica. Essa dica será disponibilizada para o GE pelo servidor da aplicação.

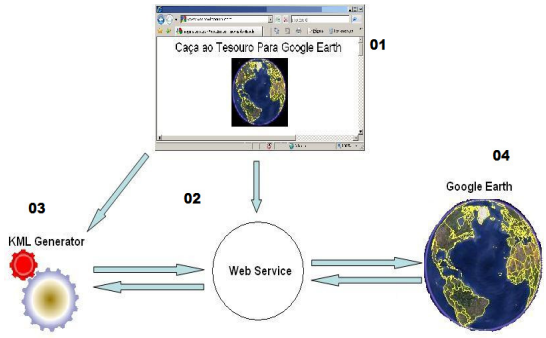


Figura 2: Funcionamento do Jogo Caça Tesouro

5. Conclusões

Aplicações como o *Google Earth* estão cada vez mais em evidência. A combinação entre entretenimento, curiosidade e conhecimento do espaço geográfico pode ser algo que atraia um público diverso, que possui objetivos distintos em relação a exploração virtual do espaço em uma única ferramenta.

Pode-se imaginar o uso do jogo Caça Tesouro como um instrumento educativo, pois o Jogador ao mesmo tempo em que aprende se diverte.

Como trabalho futuro propõe-se a adaptação do jogo Caça Tesouro para Google Earth para utilização contextualizada ao ensino de geografia, história, atualidades, ecologia, dentre outras áreas do conhecimento.

Referências

BLOWER, Jon, et. Al. Sharing and visualizing environmental data using Virtual Globes. Disponível em: http://www.allhands.org.uk/2007/proceedings/papers/792.pdf

BROWN, Martin C. Hacking Google Maps and Google Earth. Wiley Publishing, Inc. 2006.

ERLE, Shuyler e GIBSON, Rich. Google Maps Hacks. O'Reilly. 2006.

GOOGLE Inc (01). Guia do Usuário do Google Earth. 2008. http://earth.google.com/userguide/v4/ug_sharingplacedata.html

GOOGLE Inc (02). KML Tutorial. Disponível em: http://code.google.com/intl/pt-BR/apis/kml/documentation/kml_tut.html

OGC. *KML 2.2* – An OGC Best Practice, Reference number of this OGC® project document: OGC 07-113r. 2007. Disponível em: http://www.opengeospatial.org/standards/kml/

REHEM, Almerindo. KML: Linguagem de Marcação. 2008. Acessado em: 22.11.2008 Disponível em: http://www.devin.com.br/kml-linguagem-de-marcacao/

SILVERMAN, Jacob. Como Funciona o Google Earth. Traduzido por HowStuffWorks Brasil. Acessado em 25/11/2008. Disponível em: http://informatica.hsw.uol.com.br/google-earth.htm

PINA, Maria de Fátima de, SANTOS, Simone M Conceitos Básicos de Sistemas de Informação Geográfica e Cartografia Aplicados a Saúde. Brasília-DF: OPAS, 2000.

Virual Globe. Acessado em 20/06/2009. Disponível em: http://virtual-globe.org/.

Um Sistema para Geração Procedural de Terrenos Pseudo-Infinitos em Tempo-Real Utilizando GPU e CPU

Fábio Markus Miranda, Carlúcio S. Cordeiro, Luiz Chaimowicz
Departamento de Ciência da Computação
Universidade Federal de Minas Gerais

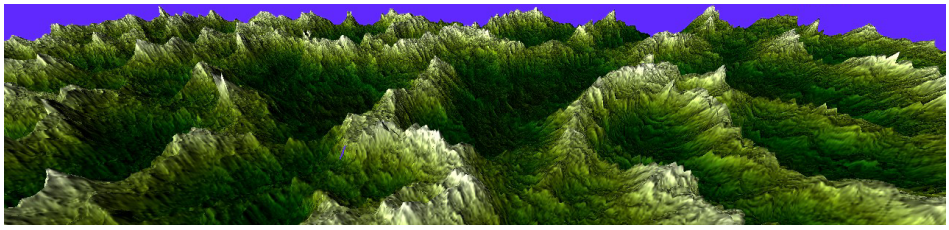


Figura 1: *Terreno gerado proceduralmente.*

Resumo

O rápido crescimento do poder de processamento das placas gráficas fez com que diversas tarefas migrassem da CPU para a GPU. Este trabalho propõe um sistema *multithread* que utiliza tanto a CPU quanto a GPU para a geração procedural de terrenos, compartilhando a carga de trabalho entre as arquiteturas. Dessa forma, busca-se minimizar o tempo gasto com a geração e assim permitir uma navegação fluida através de um mundo pseudo-infinito gerado proceduralmente.

Ao final, uma comparação é feita com base em experimentos utilizando diferentes configurações da arquitetura, com o objetivo de expor suas vantagens e aplicações.

Keywords:: computação gráfica; modelagem procedural; gpu; gpgpu; programação paralela

Author's Contact:

{fabiom, carlucio}@gmail.com
{chaimo}@dcc.ufmg.br

1 Introdução

A técnica de geração procedural é uma área da Ciência da Computação que propõe o uso de rotinas e algoritmos para a geração de algum tipo de conteúdo, seja ele um modelo tridimensional, música, animações, etc. Tal técnica vem se tornando bastante popular nos últimos tempos, considerando que, com o crescimento da indústria do entretenimento, há uma necessidade de se construir modelos cada vez maiores e com um grande nível de detalhe. Assim, técnica de geração procedural vem como uma alternativa à utilização do trabalho de artistas e modeladores na criação de modelos tridimensionais.

A área de geração procedural envolve uma pesquisa constante por algoritmos que sintetizem certos aspectos da realidade ao nosso redor, sejam terrenos [Ebert et al. 2002], cidades [Chen et al. 2008], ou vegetação [Prusinkiewicz and Lindenmayer 1996]. Paralelo a isso, há uma outra vertente de pesquisa que busca executar tais algoritmos de modo que eles possam ser utilizados em aplicações interativas de tempo-real.

Este trabalho propõe um sistema de geração procedural de terrenos que utilize tanto a CPU quanto a GPU, de maneira isolada ou combinada, com cada arquitetura ficando responsável por uma determinada porcentagem da carga de trabalho referente à geração. Através do sistema, será possível navegar em um terreno pseudo-infinito de forma interativa e sem quedas bruscas do *frame rate*.

O trabalho está organizado da seguinte maneira: na Seção 2 são apresentados alguns trabalhos relacionados e as principais contribuições deste trabalho. A Seção 3 revisa alguns conceitos pertinentes. A Seção 4 apresenta o sistema proposto para a geração procedural utilizando tanto a CPU quanto a GPU e a Seção 5 detalha sua implementação. A Seção 6 faz uma discussão sobre os experimentos realizados. Finalmente, a Seção 7 apresenta a conclusão e a perspectiva para futuros trabalhos.

2 Trabalhos relacionados

A modelagem procedural de terrenos é uma área vastamente pesquisada, com diversos trabalhos que buscam sempre criar os terrenos mais realistas possíveis.

Uma das técnicas utilizadas na modelagem procedural de terrenos é o ruído Perlin [Perlin 1985], uma função pseudo-aleatória que, dada uma entrada (posição), retorna um valor que possui uma suave transição com os seus vizinhos. Em [Perlin 2004] foi apresentado um ruído Perlin otimizado, que buscou adaptar o ruído às novas arquiteturas (GPUs), melhorar as propriedades visuais e introduzir uma única versão do ruído que retornaria os mesmos valores independentemente da plataforma de *hardware* ou *software*.

Em [Ebert et al. 2002] são apresentados alguns algoritmos que fazem uso do ruído Perlin e que são capazes de gerar terrenos de uma forma significativamente realista. Podemos citar os algoritmos *fBm*, *heterogenous terrain*, *hybrid multifractal* e *ridged multifractal*, sendo que este último foi o algoritmo utilizado neste trabalho.

Em [Cordeiro and Chaimowicz 2008], os autores apresentam um paradigma para a modelagem procedural (terrenos, vegetação, etc.) utilizando múltiplas *threads*. Uma implementação é proposta utilizando apenas as unidades de processamento disponíveis na CPU.

A geração procedural utilizando a GPU foi explorada em [Geiss 2007] e [Schneider et al. 2006]. O primeiro trabalho, faz uso de *geometry shaders* e está limitado às placas de vídeo com suporte a DirectX 10. O segundo trabalho, mais abrangente quanto as placas de vídeo suportadas, gera os terrenos na GPU com o uso de algoritmos multifractais (semelhante ao que é proposto aqui). Nenhum dos dois trabalhos, porém, faz uma comparação entre implementações de geração de terrenos utilizando a CPU e a GPU, e também não buscam uma plataforma que utilize as duas arquiteturas.

3 Conceitos básicos

A modelagem procedural de terrenos pode ser feita através da construção de mapas de altura dos terrenos. Para a geração destes mapas, as técnicas mais utilizadas são as baseadas em ruídos, como será apresentado a seguir.

3.1 Ruído Perlin

O ruído Perlin foi criado pelo Professor Ken Perlin [Perlin 1985], da *New York University* e é usado para simular estruturas naturais, como nuvens, texturas de árvores, e terrenos.

A função ruído retorna, para um dado domínio e as mesmas sementes (*seeds*), números entre 0 e 1; Em uma segunda execução, com as mesmas entradas, teremos os mesmos números entre 0 e 1. Cada valor retornado é o resultado do seguinte produto interno:

$$G \cdot (P-Q)$$

Onde P é a posição do ponto que está sendo calculado o valor do ruído, Q é a posição de um de seus vizinhos, e G é o valor de um vetor gradiente pseudo-aleatório. Os resultados do produto interno dos vizinhos é então interpolado, garantindo assim que haverá uma suave transição entre todos os valores retornados.

O resultado, como pode ser visto na Figura 2, apresenta transições suaves, diferentemente do ruído branco (puramente aleatório).

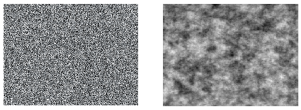


Figura 2: Esquerda: *Ruído aleatório.* Direita: *Ruído Perlin.*

As características fundamentais do ruído Perlin são a sua aparente aleatoriedade (ao menos para o olho humano); sua capacidade de ser reproduzido, dado os mesmos valores dos gradientes; e sua transição suave entre valores.

3.2 Fractais

Fractais podem ser descritos, segundo [Ebert et al. 2002], como objetos geométricos complexos, na qual a complexidade surge da repetição de uma forma em uma extensão de escalas. Um exemplo simples pode ser visto na Figura 3:

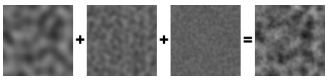


Figura 3: *Exemplo de um fractal a partir de ruído Perlin.*

Os três ruídos Perlin estão em escalas diferentes e, uma vez somados, formam um fractal, segundo a definição citada. Multifractais já são um subgrupo caracterizado pela variação de sua dimensão fractal ao longo de sua localização.

3.3 Ridged Multifractal

O *Ridged multifractal* é um modelo para geração de terrenos apresentado em [Ebert et al. 2002]. O seu principal ponto é a captura da *heterogeneidade de terrenos em grande escala*, apresentando montanhas, planaltos e crateras. Os seguintes parâmetros são levados em consideração na execução do algoritmo:

- **Octaves:** Número de iterações (e, consequentemente somas) feitas sobre a função de ruído.
- **Amplitude:** Máximo valor adicionado ao valor total do ruído.
- **Frequency:** Número de valores de ruídos definidos entre dois pontos (quanto maior a frequência, maior o disturbio da textura resultante).
- **Lacunarity:** É um termo usado no cálculo de fractais, e dita o espaço entre sucessivas frequências, aumentando ou diminuindo a densidade do resultado final.
- **Offset:** Fator multifractal.
- **Tamanho:** Tamanho do mapa de altura que será salvo o resultado da geração procedural.

O tempo de execução é dependente apenas do tamanho do mapa e do número de octaves.

4 Metodologia Proposta

O sistema proposto tem como objetivo gerar proceduralmente os terrenos tanto na GPU quanto na CPU a medida em que o usuário percorra o terreno. Portanto, foi preciso estabelecer uma base comum às duas arquiteturas.

O terreno geral é dividido em terrenos retangulares menores (chamados *patches*), como mostra o *grid* da Figura 4. Dessa forma, apenas *patches* de interesse do usuário (que estão mais próximos, por exemplo) precisarão ser gerados.

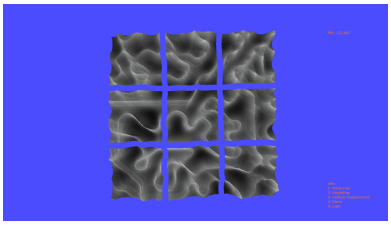


Figura 4: Patches *exibidos em um* grid.

Considerando o usuário inicialmente localizado no *patch* central, ao mover-se para um *patch* vizinho, o sistema irá requisitar a geração de novos *patches*, vizinhos a aqueles que estão na borda do grid. O número de vizinhos gerados, bem como a quantidade de vizinhos do *patch* central são variáveis do sistema, podendo ser adaptadas, pelo usuário, de acordo com o poder de processamento de sua máquina.

Para garantir uma visualização fluida do terreno, minimizando as interrupções com a geração, o sistema proposto decidirá qual arquitetura (GPU ou CPU) será utilizada na geração dos *patches* a partir de uma variável α, que representa a porcentagem de gerações que ocorrerão na *GPU*. $1 - \alpha$ representará, portanto a porcentagem de gerações na *CPU*.

A Figura 5 mostra como se dá o fluxo de geração.

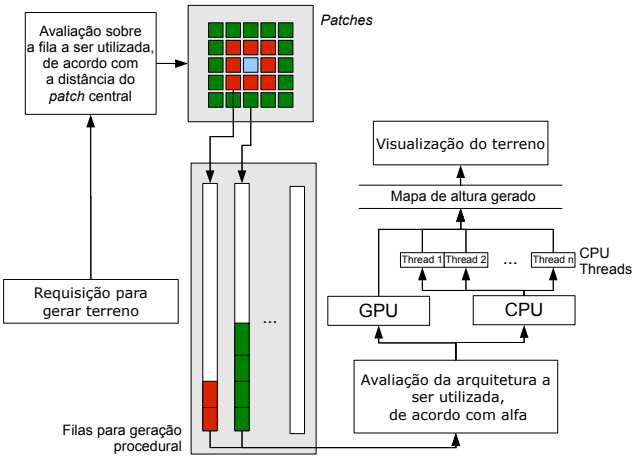


Figura 5: *Fluxo da geração procedural.*

As requisições por novos terrenos serão adicionadas a uma fila e uma política *First In, First Out* (FIFO) será utilizada para decidir qual terreno será gerado. Como é possível ver na Figura 5, o número de filas existentes no sistema será igual ao número de vizinhos do *patch* central. Dessa forma, é possível decidir quais terrenos serão gerados a partir de sua distância da câmera.

A *thread 0* (principal) ficará encarregada da requisição para gerar novos eventos, avaliação da fila, avaliação da arquitetura a ser utilizada (GPU ou CPU), e também será responsável pelas chamadas às funções OpenGL, incluindo aquelas responsáveis por iniciar a execução das instruções que serão executadas na GPU. A geração na CPU ocorrerá em outras *threads*, não a principal.

4.1 O Cálculo de α

O valor da variável α é, atualmente, controlado manualmente pelo usuário. A sua variação de acordo com a utilização de cada arquitetura será um tema a ser abordado em trabalhos futuros.

Atualmente, a maior dificuldade para medir o tempo de geração na GPU é a falta de um padrão nas extensões disponíveis em OpenGL. A extensão **GL_EXT_timer_query**, por exemplo, só está disponível em placas NVidia, algo que anularia a possibilidade da execução deste trabalho em placas ATI.

A utilização de chamadas como **glFinish()** para sincronizar a CPU e a GPU e assim medir o tempo de geração dos terrenos poderia prejudicar a performance do sistema, já que pára a execução da CPU enquanto todos os comandos OpenGL não forem executados.

Uma outra opção para a sincronização seria a extensão **GL_NV_fence**, que oferece funções para sincronização semelhantes ao **glFinish()** e **glFlush()**, porém com um grau maior de controle sobre quais comandos OpenGL deverão ser executados na chamada. Mais uma vez, porém, a extensão não está disponível para placas ATI.

4.2 Geração

Toda a geração dos terrenos na GPU é feita através de um *fragment shader* (versão 3.0), utilizando o ruído Perlin como foi proposto em [Perlin 2004]. Como toda computação de *shaders* fica limitada a geometrias ou texturas, foi preciso renderizar um quadrado utilizando as funções OpenGL, para que, dessa forma, fosse possível aplicar os *shaders* às suas primitivas e iniciar os cálculos necessários. O resultado da geração é renderizado em um *framebuffer off-screen* (que não é exibido na tela), através da extenção FBO, que permite criar novos *buffers*.

O cálculo dos vetores gradientes, necessário no ruído Perlin, é feito na CPU, apenas no início do sistema, e depois é acessado no *fragment shader* como uma textura 2D.

Como o mapa de altura é gerado na GPU, não há qualquer tipo de perda de desempenho com a transferência entre a memória RAM e a memória da placa de vídeo. Um aspecto importante é que, durante a geração do mapa de altura, os valores das normais de cada vértice também são calculados.

A geração utilizando a CPU é feita utilizando o mesmo algoritmo implementado na GPU. Como o mapa de altura gerado reside na memória principal, sua renderização dependerá da transferência para a memória da placa de vídeo.

4.3 Visualização

Com o mapa de altura gerado, o próximo passo é exibir o terreno para o usuário, que é feito de forma idêntica tanto para os terrenos gerados na GPU quanto para os gerados na CPU.

O passo inicial é a geração de uma malha (conjunto de vértices) de tamanho pré-determinado, como mostra a Figura 6. A malha é gerada de tal forma que um número maior de vértices está concentrado no centro. Quanto maior a distância, menor o número de vértices presentes. Isto propicia uma maneira rápida e fácil de implementar um algoritmo de nível de detalhe (quanto maior a distância do centro, menor será a necessidade de se renderizar o terreno em alta fidelidade).

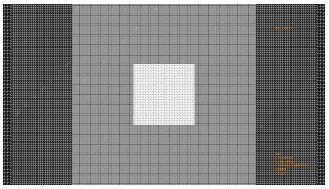


Figura 6: *Malha inicial para visualização dos terrenos.*

Como a malha é gerada apenas uma única vez (no início da execução), não é preciso criar repetidas malhas a medida que o jogador percorre o terreno. Apenas os mapas de altura de cada *patch* são trocados, como mostra a Figura 7

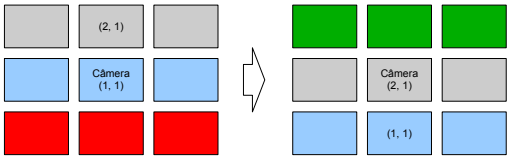


Figura 7: *Movimentação da câmera para um outro* patch.

Na Figura 7 é possível notar o deslocamento dos mapas de altura quando a câmera move para o *patch* superior ao (1,1). Para que haja uma transição, uma matriz de translação, com valores iguais ao tamanho do *patch*, é feita e multiplicada à matriz responsável por renderizar todas as primitivas, resultando na translação de todos os *patches*.

Este algoritmo de nível de detalhe se mostrou bastante eficaz para este trabalho, não sendo necessária a implementação de um algoritmo LOD mais robusto. Além disso, como sabemos o número de vértices antecipadamente, a performance do aplicativo tem uma menor chance de sofrer quedas bruscas de rendimento.

5 Implementação

A Figura 10 apresenta as camadas do sistema, bem como as bibliotecas utilizadas.

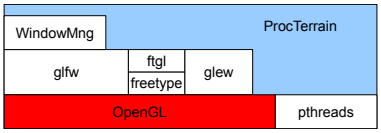


Figura 10: *Camadas do sistema.*

A camada *WindowMng* tem como propósito simular a um aplicativo gráfico genérico (*game*, simulador, etc.); desta forma, o sistema poderá ser posteriormente adaptado para funcionar em conjunto com outros aplicativos que possam ser desenvolvidos (ou acoplado a uma *engine*).

A Figura 11 apresenta em detalhes os módulos presentes nas camadas *WindowMng* e *ProcTerrain*. A seguir, uma explicação sobre cada um dos módulos.

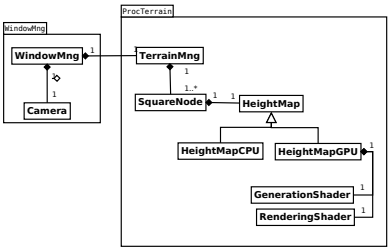


Figura 11: *Diagrama com as principais classes do sistema implementado.*

- **WindowMng**: Responsável por simular um aplicativo gráfico genérico, e chamar os devidos *callbacks* do pacote *ProcTerrain*.
- **Camera**: Módulo que implementa uma câmera controlada pelo jogador e navegando pelo mundo.
- **TerrainMng**: Módulo responsável por gerar e controlar os terrenos.
- **SquareNode**: Nodo que representa uma fatia (*patch*) do terreno.

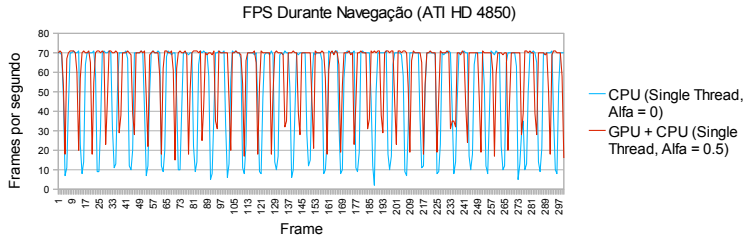


Figura 8: *Gráfico com o FPS na navegação pelo mundo durante 300 segundos, utilizando uma única* thread *na CPU.*

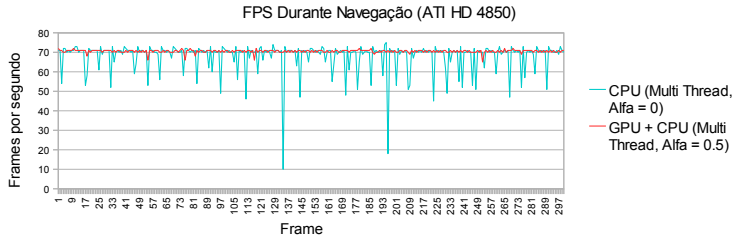


Figura 9: *Gráfico com o FPS na navegação pelo mundo durante 300 segundos, utilizando múltiplas* threads *na CPU.*

- **HeightMap**: Módulo que implementa os mapas de altura dos terrenos gerados na CPU ou na GPU. Possui os métodos **GenerateGPU** e **GenerateCPU**, que são executados em paralelo à *thread* principal.
- **GenerationShader**: *Shader* responsável pela geração dos terrenos.
- **RenderingShader**: *Shader* responsável pela renderização dos terrenos.

6 Resultados Experimentais

Para verificar a eficiência das gerações na CPU e GPU, foi feito um experimento que consiste na navegação por um trajeto constante durante 300 segundos. O computador utilizado foi um *Core 2 Duo E7400*, com 2GB de memória *RAM* e placa de vídeo *ATI Radeon HD 4850* com 512MB de memória *RAM*, e *driver* versão 8.612. As seguintes configurações foram utilizadas para a geração procedural: 8 *octaves*, *lacunarity* igual a 2.5, *gain* igual a 0.5, *offset* como 0.9, número de vizinhos referente ao *patch* central igual a 2, tamanho de textura 256. A resolução do programa foi de 1280 x 720.

Quatro experimentos foram executados, cada um com uma configuração diferente de utilização da CPU e GPU. Os dois primeiros experimentos (Figura 8) foram executados utilizando a GPU e uma única *thread* na CPU. Os experimentos três e quatro (Figura 9) utilizaram a GPU e múltiplas *threads* na CPU. A carga de geração dos terrenos foi dividida igualmente entre as duas arquiteturas em todos os experimentos ($\alpha = 0.5$).

Como mostra os experimentos exibidos na Figura 8, a geração utilizando uma *thread* na CPU possui quedas bruscas no *framerate*, com momentos em que este chega até a 2 FPS; O *framerate* médio fica em 51.79. Com a geração conjunta na GPU, as quedas são sensivelmente menores, nunca caindo abaixo dos 15 FPS, e o *framerate* médio fica em 60.94.

Utilizando uma configuração do sistema com múltiplas *threads* (Figura 9), é possível notar algumas quedas do *framerate*, embora a navegação seja consideravelmente mais fluida do que a obtida utilizando apenas uma única *thread*. O *framerate* mínimo nessa situação foi de 10 FPS, e a média durante a navegação foi de 68.08 FPS. Utilizando a geração em conjunto na GPU, o *framerate* nunca ficou abaixo de 65 FPS, e a média foi de 70.45 FPS.

7 Conclusão e Trabalhos Futuros

Os resultados obtidos na geração procedural de terrenos na GPU mostram o poder de processamento das placas gráficas em relação às CPU. A opção de se gerar nas duas arquiteturas mostra-se promissora, principalmente considerando a utilização do sistema acoplado a um *game* ou simulador. Como haverá inevitavelmente outras tarefas sendo executadas (*path-finding*, sombras, HDR), a possibilidade de se migrar a carga de trabalho envolvida na geração procedural pode ser bastante vantajosa.

Como trabalho futuro, espera-se encontrar uma estratégia eficiente para o cálculo de α. Dessa forma, a geração procedural poderá ser balanceada automaticamente entre as diferentes arquiteturas utilizadas (CPU e GPU).

O sistema foi implementado sempre tendo em mente a sua utilização acoplada a outras aplicativos. Assim, adotá-lo em um *game* ou simulador demandaria pouco esforço.

Referências

CHEN, G., ESCH, G., WONKA, P., MULLER, P., AND ZHANG, E. 2008. Interactive procedural street modeling. *ACM Trans. Graph. 27*, 3.

CORDEIRO, C. S., AND CHAIMOWICZ, L. 2008. Parallel lazy amplification: Real-time procedural modeling and rendering of multi-terabyte scenes on a single pc. *VII Brazilian Symposium on Computer Games and Digital Entertainment, 2008, Belo Horizonte. SBGames 2008*.

EBERT, D. S., MUSGRAVE, F. K., PEACHEY, D., PERLIN, K., AND WORLEY, S. 2002. *Texturing and Modeling: A Procedural Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

GEISS, R. 2007. *GPU Gems 3*. Addison-Wesley Professional, ch. 1 Generating Complex Procedural Terrains Using the GPU, 7–37.

PERLIN, K. 1985. An image synthesizer. *SIGGRAPH Comput. Graph. 19*, 3, 287–296.

PERLIN, K. 2004. Implementing improved perlin noise. In *GPU Gems*, R. Fernando, Ed. Addison Wesley Professional, March, ch. 13.

PRUSINKIEWICZ, P., AND LINDENMAYER, A. 1996. *The algorithmic beauty of plants*. Springer-Verlag New York, Inc., New York, NY, USA.

SCHNEIDER, J., BOLDTE, T., AND WESTERMANN, R. 2006. Real-time editing, synthesis, and rendering of infinite landscapes on GPUs. In *Vision, Modeling and Visualization 2006*.

Usabilidade de jogos virtuais e frequência cardíaca de usuários. Um estudo comparativo

Grassyara Tolentino[1] Alam Ventura[2] Claudio Battaglini[3]
Luciana S. Oliveira[4] Alessandra B. Matida[5] Ricardo J. de OLiveira[6]

Universidade Estadual de Montes Claros, Departamento de Educação Física, Brasil. [1]
Universidade Católica de Brasília, Brasil[2,4,5,6]
University of North Carolina at Chapel Hill, North Carolina, USA[3]

Resumo

Serious games são aqueles que têm por objetivo informar, alertar, educar e conscientizar. Mantendo, ainda, as características próprias dos jogos virtuais de entretenimento, como por exemplo, a diversão e o desafio. No entanto, alguns estudos apontam limitações tecnológicas destes softwares que incidem diretamente sobre sua qualidade, reduzindo a satisfação dos jogadores e seu potencial como ferramenta educacional, por apresentarem-se monótonos e sem atrativos. As emoções geradas pelos jogos virtuais têm sido analisadas por alguns estudos, apontando uma associação entre as emoções e o sucesso dos games. Parâmetros fisiológicos são uma forma objetiva de mensurar alterações emocionais em usuários de jogos virtuais. Baseado no exposto, este estudo teve como objetivo avaliar a usabilidade e a freqüência cardíaca entre um serious games e um jogo virtual de entretenimento.

Palavras-chave: jogos virtuais, usabilidade, parâmetros cardiovasculares

Authors' contact:
{grassyara2, lubaiucha4 gessymaratolentino5}
@yahoo.com.br
{claudio3}@email.unc.edu
{alamc1, amatida5, rjaco}@ucb.br

1. Introdução

Serious games são jogos, ou softwares, cujo objetivo é combinar características lúdicas dos jogos virtuais a aspectos sérios como o ensino, a aprendizagem, a comunicação e ainda a informação, não sendo essas classes exaustivas [Alvarez 2007]. O caráter formativo/educacional é que diferencia os jogos sérios dos jogos virtuais de entretenimento que podem ser aplicados ao ensino, sendo, contudo sua gênese voltada apenas para o lúdico [Zyda 2005].

No entanto, tomar parte no mundo das "coisas sérias" não é uma tarefa nada fácil, uma vez que os jogos virtuais geralmente têm uma imagem social controversa, estando associado à perda de tempo, baixo rendimento escolar, problemas sociais como isolamento, agressividade, e até mesmo a problemas de saúde como a obesidade [Anderson et al. 2008; Abreu et al. 2008; Carnegey and Anderson 2005].

Savi and Ulbricht [2008] vão mais além quando afirmam que os jogos sérios enfrentam dificuldades em relação aos objetivos de aprendizagem e artefatos pouco divertidos face aos games convencionais. Estes fatos poderiam limitar a aceitação e a satisfação do usuário quanto à utilização dos jogos sérios. A usabilidade é um dos requisitos à qualidade dos softwares, sendo que a má usabilidade prejudica as experiências dos usuários, e consequentemente a diversão e o aprendizado [Gurgel et al. 2006].

De acordo com Rajava et al. [2005], as emoções englobam três componentes distintos: a experiência subjetiva, o comportamento expressivo, e o componente fisiológico. Logo, as vivências com jogos virtuais teriam a capacidade de alterar os sistemas biológicos relacionados à emoção, como o sistema nervoso central (SNC). Estes dados são corroborados por alguns estudos que têm descrito alterações em parâmetros fisiológicos no período de jogo, em variáveis como: consumo de oxigênio, atividade muscular, alterações galvânicas da pele, pressão arterial, frequência cardíaca e gasto calórico [Tolentino 2009; Borusiak et al. 2008; Wang and Perry 2006].

Com base em dicotomias hipotéticas sobre a usabilidade e fatores emocionais como diversão e desafio entre jogos virtuais sérios e de entretenimento, emergem alguns questionamentos. Será que usuários de jogos de entretenimento têm uma avaliação favorável quando utilizam jogos sérios? Será que esses games proporcionam alterações fisiológicas compatíveis com a realização da tarefa nos jogadores? Será que a usabilidade está relacionada a alterações cardiovasculares ocorridas durante o jogo?

A partir do exposto, o presente estudo teve como objetivo mensurar e correlacionar a satisfação subjetiva do usuário (SUS), frequência cardíaca (FC) de jogadores durante um período de tempo entre um jogo sério e um jogo de entretenimento.

2. Materiais e Métodos

Delineamento de estudo: O presente estudo caracterizou-se como quantitativo e transversal [Marconi and Lakatos 2003].

Amostra: A pesquisa foi realizada em Taguatinga-DF com a amostra constituída de oito voluntários com idade acima de 18 anos, experiência em jogos por computador, não-fumantes ou usuários de drogas psicoativas e que não apresentavam condição crônico-degenerativa.

Softwares: Os softwares utilizados nesta pesquisa foram: a) *Re-mission* (*HopeLab Foundation*, Redwood City, CA, EUA) um *serious games* de aventura, *single-player*, com um roteiro pré-determinado, e tiro em 3ª pessoa, cujo objetivo é derrotar células cancerosas dentro do corpo humano e ensinar o usuário a lidar com os efeitos colaterais do tratamento antineoplásico; b) *Counter-Strike: Source (Valve,* Bellevue, WA, EUA) um massively multiplayer *on line* game (MMOG), caracterizado como jogo de simulação de guerra urbana e tiro em 1ª pessoa, onde os objetivos giram em torno de ações militares como salvar reféns, eliminar terroristas, desarmar bombas, dentre outros.

Hardwares: O laboratório de informática foi equipado com: 10 desktops ITAUTEC® Infoway CCTZ (Processador Intel® Pentium IV); 10 fones de ouvido estéreos da marca NEOX® e modelo FC308MV; Switch D-Link® DES-1024D. A rede utilizada para a realização dos testes foi isolada logicamente.

Questionário de Avaliação da Satisfação Subjetiva do Usuário (*System Usability Scale* - SUS): A satisfação subjetiva do usuário foi avaliada através do SUS, que é uma escala de usabilidade de 10 itens, desenvolvida em 1986, por John Brooke [1996], onde as respostas são marcadas numa escala de Likert, e os escores finais podem variar de 0 a100. Quanto maior o valor mais favorável será a usabilidade percebida pelo usuário. O SUS tem sido disponibilizado gratuitamente para uso em pesquisas envolvendo a sondagem da opinião do usuário, sendo solicitada apenas a devida referência à fonte de mensuração, i.e., à Digital Equipment Co. Ltd. (DEC), Reading (UK).

Frequência cardíaca: A Frequência Cardíaca (FC) foi mensurada, utilizando-se um monitor de FC, modelo (T71) da *Polar® Sport Tester*. Essa variável foi aferida na posição sentada, em repouso, e a cada 1 minuto após o início de cada jogo, totalizando 20 aferições.

Procedimentos A coleta de dados foi realizada em três etapas distintas. Na primeira, foram explicados todos os procedimentos e objetivos da pesquisa aos jogadores e solicitado a eles que assinassem o Termo de Consentimento Livre e Esclarecido. A seguir, foi feita uma ambientação dos jogadores à sala de coleta, aos jogos e a adaptação dos pesquisadores às medidas biológicas nos indivíduos sentados e em condições de jogo. Na segunda etapa, os voluntários jogaram o *Counter-Strike*. Inicialmente foi solicitado que os jogadores permanecessem sentados em sua base de jogos por 3 minutos, já conectados aos aparelhos de avaliação fisiológicos. Após este período, foi aferida a FC de repouso. Então o pesquisador responsável deu o sinal de um minuto, solicitou aos voluntários que colocassem os fones de ouvido, a luz do ambiente foi reduzida e o jogo iniciado. Assim que se iniciou o jogo, foi aferida a FC e, em todo minuto subsequente, até completar 20 minutos. A terceira etapa foi realizada da mesma forma que a segunda, contudo o jogo era o *Re-Mission*. Antes de começar cada coleta de dados, foi oferecido aos voluntários um lanche, contendo carboidratos, proteínas e um líquido. A sala reservada à coleta foi mantida em temperatura controlada.

Análises estatísticas: A normalidade dos dados foi testada através do teste de *Shapiro-Wilk*. A diferença entre as variáveis dependentes foi testada através do teste de Wilcoxon, e do teste qui-quadrado. Nas análises das correlações, utilizou-se o ρ de Spearman. O software estatístico foi o Statistical Package for the Social Sciences (SPSS®), versão 14 para Windows®. Foi adotado o nível de significância $p \leq 0,05$.

3. Resultados

Os dados de caracterização da amostra informaram que a maioria dos voluntários, 62,5%, declararam ter bom conhecimento sobre Tecnologia da Informação e 37,5% informaram ter um ótimo conhecimento. Cerca de 87,5% dos voluntários afirmaram possuir mais de 10 anos de experiência em jogos por computador. Além disso, 37,5% da amostra declararam-se experientes no *Re-mission* e 75%, no CS. A maioria possuía curso superior incompleto (87,5%).

A tabela 1 apresenta a média de idade dos participantes e os resultados do SUS de cada jogo. De acordo com essa tabela, não foram identificadas diferenças estatisticamente significativas entre o CS e o *Re-mission*,

Tabela 1: Valores médios ($\overline{x}$), desvios-padrão (SD) mínimos e máximos da idade dos usuários, usabilidade e escores do *Re-mission* e CS.

Variáveis	$\overline{x}$ ± SD	Mínimo	Máximo
Age (years)	25,67±2,83	22,00	29,00
SUS_*Re-mission*	63,85±22,20	22,50	87,50
SUS_CS	80,00±11,01	60,00	97,50

SUS = System Usability Scale; RM=*Re-mission*; CS = *Counter Strike*

A análise dos dados fisiológicos apresentados na tabela 2 revelou efeito significante na frequência cardíaca média entre os dois jogos (p=0,017). Não foram identificadas associações estatisticamente significantes entre a usabilidade e a frequência cardíaca.

Tabela 2: Valores médios ($\overline{x}$), desvios-padrão (SD) da Frequência cardíaca, pré, inicial e durante a fase dos jogos.

Variáveis	Frequência Cardíaca (FC)		
	Repouso	Inicial	Média
	$\overline{x}$ ± SD	$\overline{x}$ ± SD	$\overline{x}$ ± SD
Re-mission	77,75 ± 9,3	78,62 ± 7,8	79,54 ± 6,6*
CS	83,59 ± 8,3	83,87 ± 9,5	86,90 ± 1,7

RM=*Re-mission*; CS = *Counter Strike;* * efeito significante observado na variável média da FC entre o *Re-mission* e o CS (p=0,017).

4. Discussão dos Resultados

As análises informaram que os dois games avaliados, embora projetados com finalidades diferentes, obtiveram valores similares de satisfação subjetiva do usuário. Este é um aspecto extremamente favorável aos serious games, uma vez que se opõe aos estudos que afirmam que os jogos com finalidades educativas podem apresentar limitações que reduzem a qualidade do software [Savi and Ulbricht 2008; Hack et al. 2002].

O CS é um MMOG (massively multiplayer online game) de entretenimento comercializado no mundo inteiro e que apresenta várias versões [Jansz and Tanis 2007], logo, hipotetiza-se que diversos testes de usabilidade e melhoramentos tenham sido realizados, favorecendo cada vez mais a satisfação do usuário com este sistema, uma vez que o consumo destes games está diretamente associado à satisfação do usuário. Sendo o *Re-mission* um serious games não comercializado, e destinado a auxiliar os jogadores a lidar com tratamento antneoplásico, é de grande valia que o sistema apresente-se amigável ao usuário, visto que a proposta do game é impedir que o adversário (o câncer) vença. Logo, além da habilidade específica para o manuseio de jogos por computador, a usabilidade do software deve ser a mais favorável possível, uma vez que esta pode interferir no desempenho do jogador. Filho [2005] corrobora com essas idéias, afirmando que um jogo eletrônico, independente da plataforma ou suporte, deve apresentar uma interface que maximize a intuitividade do jogador e apresente funcionalidade, legibilidade, leiturabilidade, produtividade, e uma estética adequada ao tema e/ou grupo utilizador. O SUS é um instrumento que traz indicações sobre intuitividade, otimização entre a facilidade e a complexidade do game, e ainda bem-estar ao utilizar o software. O que sugere que o *Re-mission* apresenta requisitos ergonômicos que lhe garantem elevada qualidade, de acordo com Normas Internacionais [*ISO Norm* 9241 e *ISO Norm* 9126]

No tocante a FC, percebeu-se que os jogadores apresentaram uma elevação significativa da FC ao utilizarem o CS quando comparado ao *Re-mission*. Este dado está de acordo com estudos que avaliaram parâmetros fisiológicos durante a utilização de jogos por computador [Tolentino et al. 2009; Arriaga et al. 2008; Sharma et al. 2006; Borusiak et al. 2008]. A FC é um parâmetro fisiológico diretamente relacionado ao estresse físico e mental. E pode ser alterado por estímulos sonoros, visuais, cinestésicos e sociais; bem como processos emocionais e cognitivos [Loures et al. 2002]. Alguns trabalhos propõem que jogos virtuais com temas violentos podem conduzir a alterações cardiovasculares [Barlett 2009; Arriaga et al. 2008], no entanto os dois jogos utilizados nesse estudo envolviam armas, tiro, inimigos e outros signos relacionados à violência. Desta forma, possivelmente este não seria um aspecto determinante para as alterações cardiovasculares.

Entretanto, conjectura-se que as alterações na FC advieram de características intrínsecas a um MMOG que, possivelmente, conduziram a um maior senso de presença nos jogadores. Senso de presença é definido como a capacidade de conectar-se ao jogo, ou seja, uma percepção ilusória de não-mediação entre o homem-jogo [Rajava et al. 2004]. E, de acordo com este autor, jogos com maior senso de presença teriam maior possibilidade de alterar as variáveis fisiológicas. Entretanto, o presente estudo não possui *desing* adequado para estabelecer relações de causa-efeito.

Características como: tiro em primeira pessoa; simulação de um ambiente real de guerra; manipulação do armamento bélico, presença de missões, mapas, avatares [Clarke and Duimering 2006]; não-linearidade do jogo, manifestação da colaboração e/ou competição entre jogadores [Manninen 2001]; canais de comunicação em tempo real [Wood et al. 2004] e a presença de times virtuais com metas antagônicas [Haslher and Koch 2004]; podem ter desencadeado um maior senso de presença nos jogadores durante o CS, favorecendo os desafios e desencadeando respostas emocionais mais acentuadas nos jogadores. O que não foi percebido no *Re-mission*.

5. Conclusões

O presente trabalho concluiu que os usuários avaliados apresentaram satisfação subjetiva similar tanto no serious games *Re-mission* como no jogo de entretenimento *Counter Strike*. Os dois games induziram a reatividade cardiovascular nos usuários, sendo que o CS produziu elevação significativa na FC durante o jogo quando comparado ao *Re-mission*. No entanto, as alterações ocorridas não se apresentaram clinicamente relevantes, e todos os indivíduos mantiveram-se numa faixa de conforto fisiológico. Não foram encontradas correlações entre usabilidade e parâmetros fisiológicos. As limitações do presente estudo encontram-se no tamanho amostral e controle de variáveis como aprendizado e rejogabilidade. Seus pontos positivos seriam: a originalidade, a utilização de medidas fisiológicas para avaliar o estresse, e a comparação entre jogos de gêneros diferentes. Novos estudos são necessários para maiores esclarecimentos sobre a temática.

Agradecimentos

Os autores agradecem ao Hopelab Fundantion por ceder o software *Re-mission* para a utilização em pesquisas, e à Digital Equipment Co. Ltd. (DEC), Reading (UK) por gentilmente ceder a System Usability Scale para a realização do presente estudo

Referências

ABREU, C.N. et al., 2008. Dependência de Internet e de jogos eletrônicos: uma revisão. *Rev Bras Psiquiatr*, 30(2).156-67.

ALVAREZ, J., 2007. *Du Jeu Vidéo Au Serious Game. Approches culturelle, pragmatique* et *formelle.* Tese. Université TOULOUSE II. Toulouse le Mirail.

ANDERSON, C. A. et al., 2008. Longitudinal Effects of Violent Video Games on Aggression in Japan and the United States. *Pediatrics*.v.122, n.5, 2008.

ARRIAGA, P., et al., 2008. Are the Effects of Unreal Violent Video Games Pronounced When Playing With a Virtual Reality System. *Aggressive Behavior*, 34,521–38.

BARLETT, C., et al., 2009. How Long Do the Short-Term Violent Video Game Effects Last? *Aggressive Behavior*, 35, 225–36.

BORUSIAK, P, et al., 2008.Cardiovascular effects in adolescents while they are playing video games: A potential health risk factor? *Psychophysiology***,** 45, 327–32.

BROOKE, J. 1996. SUS: A "quick and dirty" usability scale. In: Jordan, P. W., Thomas, B., Weerdmeester, B. A., McClelland (eds.) Usability Evaluation in Industry pp. 189--94. Taylor & Francis, London, UK.

CARNAGEY, N.L.; ANDERSON, C.A., 2005.The effects of reward and punishment in violent video games on aggressive affect, cognition, and behavior. *Psychol Sci,*16(11), :882-9.

CLARKE, D.and DUIMERING, P.R., 2006. How Computer Gamers Experience the Game Situation: A Behavioral Study. *ACM Computers in Entertainment,* 4(3),1-23.

FILHO, A.M.S.; 2008. Usabilidade: o usuário tem a última palavra sempre e determina o sucesso ou não dos produtos. *Revista Espaço Acadêmico,* 82, Avaliable from: http://www.espacoacademico.com.br/082/82amsf.htm. [Accessed 30 June 2009].

GURGEL I.; et al.; 2006. A Importância de Avaliar a Usabilidade dos Jogos: A Experiência do Virtual Team. *In: SBGAMES - Anais do Simpósio Brasileiro de Jogos de Computador e Entretenimento Digital***.** nov. 22-99 : Recife – PE.

HAHSLER, M. and KOCH, S.; 2004. Cooperation and disruptive behaviour - learning from a multi-player internet gaming community. *In Piet Kommers, Pedro Isaias, and Miguel Baptista* Nunes, editors, *IADIS International Conference Web Based Communities.* Lisboa, Portugal, 35-42.

INTERNATIONAL STANDARDS ORGANIZATION, ISO NORM 9126. *Software Poduct Evaluation - Quality Characteristics and Guideline for their Use*, 1991
.

INTERNATIONAL STANDARDS ORGANIZATION, ISO NORM 9241. *Ergonomic Requirements for Office Work With Visual Display Terminals,* 1993.

JANSZ, J. and TANIS, M.; 2007. Appeal of playing online First Person Shooter Games *Cyberpsychol Behav*,10(1):133-6.

LOURES, D.L.; et al.; 2002. Estresse Mental e Sistema Cardiovascular. *Arq Bras Cardiol,* 78 (5), 525-30.

MANNINEN, T.; 2001. Virtual Team Interactions in Networked Multimedia Games - Case: "Counter-Strike" – Multi-player 3D Action Game. *In Proceedings of Presence 2001 Conference*, Philadelphia, USA, p: 1-9

MARCONI, M.A. and LAKATOS, E.M.; 2003. *Fundamentos da Metodologia Científica.* 5º ed. São Paulo: Atlas.

RAVAJA, N., et al. 2004. The psychophysiology of video gaming: Phasic Emotional Responses to Game Events. Available from: http://www.digra.org/dl/search_results?general_search_index=authors. .[Accessed 19 June 2009.

RAVAJA, N., et al. 2004. *Emotional Response Patterns and Sense of Presence during Video Games:* Potential Criterion Variables for Game Design. 2004. Available from: http://portal.acm.org/citation.cfm?id=1028014.1028068. [Accessed 12 December 2006].

SAVI, R. and ULBRICHT. V.R.; 2008.Jogos digitais educacionais: benefícios e desafios. *Novas Tecnologias na Educação*, 6(2).

SHARMA, R.; et al. Assessment of Computer Game as a Psychological Stressor. *Indian J Physiol Pharmacol.* v. 50, v.4, p: 367–374, 2006

SILVA, C.E.M.S. 2008. Experiência com jogos digitais e causas sérias. *Contemporânea,* 11 (2), 74-85.

TOLENTINO, G.P.; et al., 2009. Cardiovascular parameters comparison between single and multi-players virtual gamers. *In: 6 CONTECSI International Conference on Information Systems and Technology Management,* 2009, São Paulo. 2552-74.

WANG, X.; PERRY, A.C.; 2006. Metabolic and physiologic responses to video game play in 7-to 10-year-old boys. *Arch pediatr adolesc med*,160:411-15.

WOOD, R. T. A.; 2004. The Structural Characteristics of Video Games: A Psycho-Structural Analysis. *Cyberpsychology & Behavior,* 7(1), 1-11,

ZYDA, M.; 2005. *From Visual Simulation to Virtual Reality to Games*, IEEE Computer Society, 2005. available from: http://gamepipe.usc.edu/~zyda/pubs/Zyda-IEEE-Computer-Sept2005.pdf. [Accessed 25 January 2009].