Using Artificial Intelligence in Computational Games

Carlos A. B. C. W. Madsen, Diana F. Adamatti
Centro de Ciências Computacionais
Universidade Federal do Rio Grande
Brazil
carlos.madsen, dianaadamatti@furg.br

ABSTRACT: *This paper presents the conceptual definition of a framework to help in the construction of intelligent games, where Artificial Intelligence Techniques could be inserted in the game in an easier way. The requirements analysis of the main AI techniques are presented as well as the preliminary results of this framework.*

Keywords: Artificial Intelligence Techniques, Computer Games

Received: 4 January 2011, Revised 18 February 2011, Accepted 24 February 2011

1. Introduction

Working with computational games has a enormous motivation. In Brazil, for example, the experience of the game industry in world has shown extremely positive, where some companies have moved more than $ 30 billion, including distribution, packaging, marketing and advertising.

The application of games in education is extremely positive, because the games could help in learning process, because they develop skills such as memory, attention, creativity and reasoning. Games are no longer part just of the entertainment. A new area of research, called serious games, where it works in important themes, as natural resources management, political problems, business scenarios, etc., is growing because the researchers and the industry realized that has great potential.

In literature, there are several definitions for games. In the context of this paper, a good definition is "Game: a voluntary activity or occupation (which may be physical or mental), exercised within certain time and space, according to rules freely created, but absolutely mandatory, which has an end in itself, and whose purpose is recreation"[2]. The games can be classified in several ways, but in computer games they are a simulation of some mechanism or situation that may or may not exist in the real world, from a computational device. The computer games differ from other software for their subjectivity, because their main goal is fun (it can be associated to other goals, such as teaching, training, etc.). Even though the game is based on a real and deterministic game, there is subjectivity, since the graphical interface and the level of the game's features are based on its creator and it is adjusted to a user audience [2].

Artificial Intelligence (AI) seeks to develop tools that assist in the implementation of human activities [10]. In practice, the main goal of AI is to help the machines to perform tasks that were previously performed only by people. The computer games and AI areas have a long tradition of cooperation. The games provide the AI a variety of problems to be solved and the great challenge is the answers in real time, as MMORPGs (Massive Multiplayer Online Role-Playing Games), where many players interact each other via web. The AI gives to games more realistic features for interaction with users, such as the unpredictability of behavior [10]. Laird and van Lent [4] present several types of computer games that using AI, with dynamic roles. Genesereth et al. [3] present a new dynamic development of computer games, entirely based on AI, called General Game Playing (GGP) that tries to develop any types of games from a declarative description.

Normally, the focus of the computer games is the graphical interfaces. However, it is necessary to provide smart games, making the relations between users and system more dynamics. This paper proposes a conceptual definition for a framework to help in the construction of intelligent games, where AI techniques (intelligent agents, fuzzy logic, recommender systems, genetic algorithms, etc.) could be inserted in the game in an easier way. This framework should provide to developers a skeleton of the players and environment of proposed game, using AI techniques.

This paper is organized in 5 sections. In Section 2 is presented the conceptual proposed framework, basing the UML diagrams. Section 3 presents the requirements analysis of the chosen AI techniques that they will be used. Section 4 presents the first results with the CUDA architecture and section 5 the conclusions and the further work of the paper.

2. Conceptual Definition of the Proposed Framework

The main objective of this research is developing a tool to help in development of Computer Games using AI techniques. Therefore, we have defined the architecture for the proposed framework, divided in four modules: AI Techniques, Graphical Interface, Parser and Pre-source Code. The first module (AI Techniques) is used to the subsequent module (Graphical Interface) and so on, as presented in Figure 1.

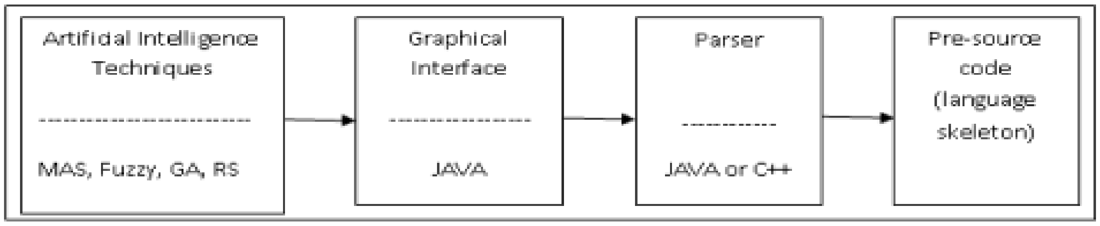
Artificial Intelligence Techniques
MAS, Fuzzy, GA, RS
Graphical Interface
JAVA
Parser
JAVA or C++
Pre-source code (language skeleton)

Figure 1. The architecture for the proposed framework

For each module, some requirements are defined:

AI Techniques module: it has all logic data structures for each AI technique and it must provide the parameters for each technique to the Graphical Interface module.

Graphical Interface: it has all graphical objects to use AI techniques in the reasoning of the players and the environment of the game. For each technique, different structures could be used. The idea is similar to CASE tools, in Software Engineering area, where for each different diagram (class, sequence, etc), there are different graphical objects.

Parser: it is a translator to the graphical definition to a computational language. The first idea is to transform to Java and C++ languages.

Pre-source Code: it is the final result of the framework. It will provide a presource code file, with the skeleton of the reasoning of the player and the environment, using AI techniques. This file will be incorporated in the final game source code.

The AI techniques module is the most important in this work. Initially, we have chosen to define three AI techniques:

Fuzzy Logic: can be defined as a logic that supports approximate reasoning rather than exact and it is based on fuzzy set theory [11]. This logic was developed by Lotfi A. Zadeh University of California at Berkeley in the 60s and it combines multi-valued logic, probability theory, artificial intelligence and neural networks that can represent human thought. The truth value of a proposition can be a fuzzy subset of any partially ordered set, unlike the binary logic systems, where the true value can only take 2 values: true (1) or false (0) [11].

Genetic Algorithms (GA): are a family of computational models inspired by evolution. These algorithm model a specific problem using a data structure similar to a chromosome [9]. An implementation of genetic algorithm starts with a population, usually random, of chromosomes. These structures are programmed to generate solutions of the problem and chromosomes with the better solutions will reproduce faster than chromosomes with the worse solutions.

Neural Networks: are inspired by the way biological nervous systems, such as the brain, process information. The key element of this paradigm is the novel structure of the information processing system. It is composed of a large number of highly interconnected processing elements (neurones) working in unison to solve specific problems. Neural Networks, like people, learn by example. Learning in biological systems involves adjustments to the synaptic connections that exist between the neurones [12].

The class diagram in UML of the proposed framework is presented in Figure 2. In this class diagram, we are defined the following classes:

Environment: this class represents the game, where the agents (players) can move and interact with each other.

Link: one environment can be connect a other environment by a "Link". This class represents the geographic position of the current environment and a pointer to the new environment. For example, the current environment of the agent is a village and each building of this village has a Link to its internal environment.

Agent: this class represents the agent (the player) with its attributes, its status and its behaviors in each situation of the game.

Character: this class is a specialization of agent class, that represents all characters into the environment.

Scenario: this class is a specialization of agent class, that represents all static objects into the environment, as trees, rocks, etc.

Attribute: this class stores each specific characteristic of an agent, like the class attributes. For example: weight (Float), height (Float), name (String), energy (Float), and so on.

State: this class represents the state set of an agent, like dead, sleeping, running, anger, happy, and so on.

Behavior: this class represents the agent behavior when it finds other agents and it must take a decision and it executes an action. To decision making, the agent analyzes the status and attributes of other agents, which feed the AI techniques. For example, if an agent uses the SOM (Self Organization Map) technique as AI technique, it could classify other agents as friends or opponents. In the case of opponent, the process could be transferred to a fuzzy logic tecnhique to classify this agent as "very strong", "strong", "medium" or "weak", before it decide to fight or not

AITechnique: this abstract class represents the AI techniques, receiving as parameter an object of class Agent.

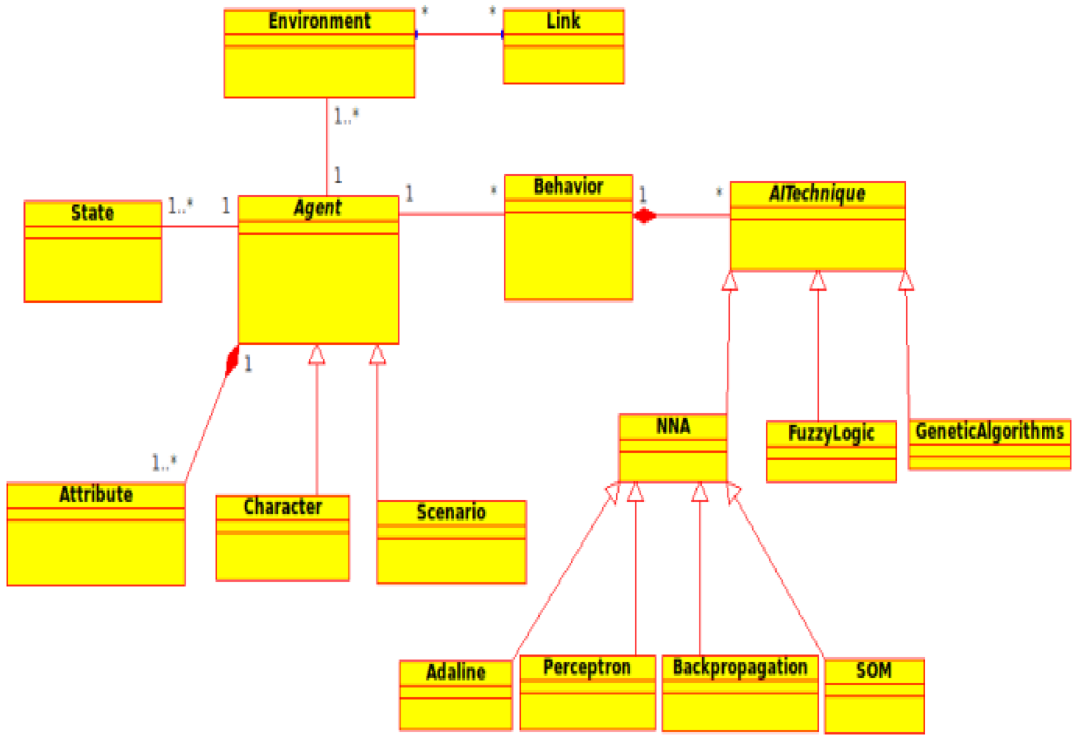

Figure 2. The class diagram in UML

3. Requirements Analysis

In Software Engineering, requirements analysis encompasses those tasks that go into determining the needs or conditions to meet for a new or altered product, taking account of the possibly conflicting requirements of the various stakeholders, such as beneficiaries or users. It is critical to the success of a development project and all requirements must be documented, actionable, measurable, testable, related to identified business needs or opportunities, and defined to a level of detail sufficient for system design [13].

To have a better system, more complete and efficient, we did the requirements analysis for each AI technique. In this requirements, we defined the initial variables and steps of execute, presented below.

a. Genetic Algorithms:

Initial variables:

- quantity of chromosomes
- size of population
- fitness function
- stop condition (by fitness function or by number of algorithm executions);
- crossover rate;
- mutation rate;

Initialization of variable:

- population by a random function;

Steps:

1. execute the algorithm with the initial population;
2. test the stop condition (fitness function or number of executions);
3. choose the better chromosomes, basing to fitness function (normally a rate of population, like 20 or 30%);
4. create new chromosomes basing crossover and mutation rates of the better beings;
5. execute the algorithm with the new population;
6. return to step 2.

b. Artificial Neural Networks - ANN (Feedforward with Backpropagation Training Algorithm):

NNA - Initial variables:

1. activation function (linear or sigmoid[1]);
2. number of hidden layers, in this example two;
3. number of neurons in the entrance layer ;
4. number of neurons in the hidden layer;
5. number of neurons in the output layer ;
6. initial weights between the entrance and hidden layers;
7. initial weights between the hidden and output layers;

Training - Initial variables:

- learning rate;
- momentum;
- acceptable error;
- number of repetitions;

[1]The backpropagation algorithm was originally developed to use the sigmoid function, but nowadays the linear function is also used in prevision problems

NNA - Initialization of variable:

- weights between the layers with a random function;

Training - Initialization of variable:

- learning rate;
- momentum;
- acceptable error;
- training the NNA with sample cases;

Steps:

1. set the value in each entrance neuron;
2. execute the ANN propagation;
3. assess if the result was positive or not (for sample, the enemy was hit);
4. training the ANN with the previous result;
5. return to step 1.

4. Preliminary Results using CUDA

Computer Unified Device Architecture **(**CUDA) is a parallel computing architecture for general purpose developed by NVIDIA®, aimed to the Graphic Processing Unit (GPU) available by this manufacturer to obtain high degree of parallel processem [15]. Originally, CUDA focus was graphic processem. Nowadays, it is used in several applications, known as General-Purpose Computation on Graphics Processing Units (GPGPU), as computational geoscience, chemistry, medicine, modeling, biology, finance and so on. [14]. In this paper, the CUDA technology will apply to parallelization of AI algorithms, which will be encapsulated in the framework classes and the programmer can abstract the details of its implementation.

4.1 CUDA Programming Model

The CUDA Programming Model can be considered of low level, because the programmer must have a good knowledge of its architecture in order to efficiently program it. There are two devices that can be programmed, the host, which is the computer hardware, and the computer card device GPU NVIDIA®. The source codes are processem by a device called kernels. Each kernel is executed under an array of parallel threads.

When the host demands a kernel to be executed in the device, it is a grid of blocks, where each block is constituted by n threads. Its ocos into the internal organization of the GPU, which is composed by n processors that can execute a limited number of parallel threads (see Figure 3). To have the best performance, the number of kernel blocks and the number of processors from the GPU should be the same, as well as the number of threads from each block and the number of threads supported by theprocessor. If there are more blocks than processors or more threads for each block than supported by the processors, it will be necessary scheduling, causem a performance reduction. Otherwise, the difference of threads will be executed with no effect, causing a processing waste.

Besides this information, the programmer should also worry about the correct use of the memory, as shown in Figure 4. There are three[2] main types of memories: local memory, shared memory and global memory.

The local memory is where the internal variables of threads are located. It is very fast and hás the same access time as a register. However, its size is much small. The shared memory itself is the memory belonging to a whole block. All the threads from this block can share information from this memory, with an access time similar to the local memory. Finally, the global memory is the computer card memory itself, and the only one which can accessed by the host. It hás a large size and low speed, comparem to two other memories. It is very important to programmer uses these memories, in order to reach an adequate performance.

4.2 An example of CUDA usage in AI

As an example of CUDA usage in the AI techniques will be presented as a parallel programming of an artificial Adaline neuron,

[2]Overall, there are six different types of memories in the CUDA technology, which are: registers, local memory, shared memory, global memory, constant memory and texture memory [14].

Weir the activation function considering the entrances by the weights, as it is shown in Figure 5 [10].

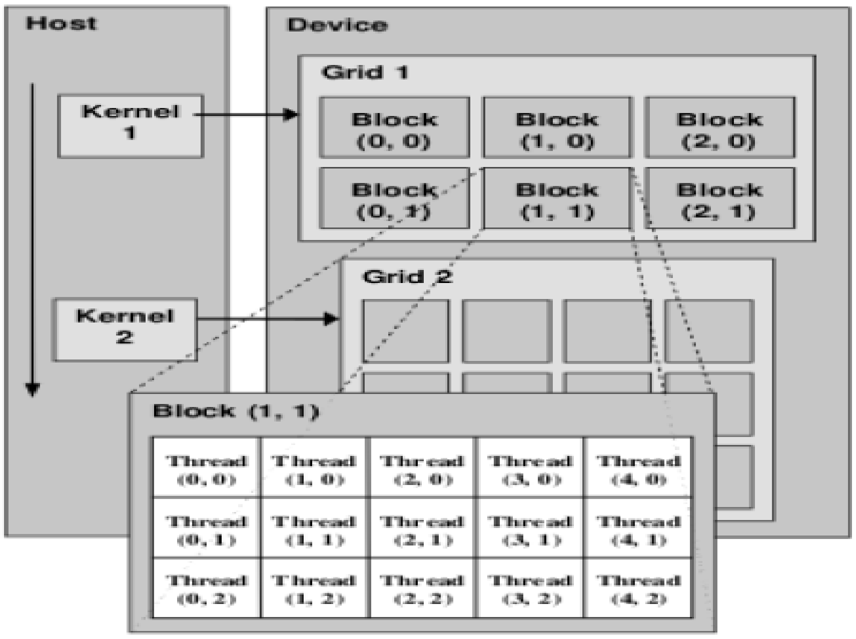

Figure 3. CUDA Programming Model [14]

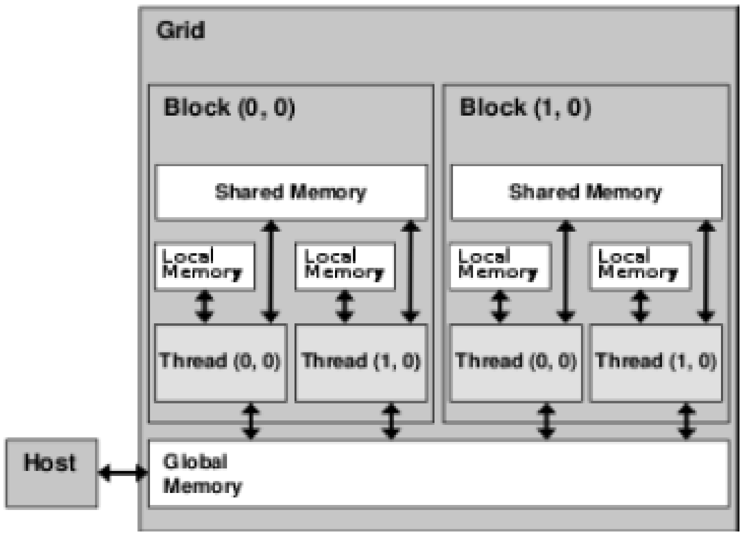

Figure 4. Kernel memory access [14]

$$net = \sum_{i=1}^{n} xi\, wi$$

In a sequential solution, written in the programming language C, part of a code for this problem is presented in Figure 6.

Figure 7 presents the implementation using the CUDA technology. Initially it should be highlighted the kernel configuration (line 26) defined by the single-block grid, compuser by an array of 32 threads (line 27). Followed by the “propagation” kernel that will be executed in the device (line 28). This execution will occur with the parameters “cuda_x” (array with neuron entrances), “cuda_w” (array with the weights referring to the entrances) and “cuda_net” (propagation return vector). All information were sent to the device global memory (lines 22,23 and 24).

```
01  int inputs=32;
02  double net=0.0;
03  form (int i=0;i<inputs;i++) {
04           net += x[i]*w[i];
05  }
```

Figure 6. Sequential Implementation in C

```
01  __global__ void propagation (double *x,double *w, double *net) {
02      int i = blockDim.x * blockIdx.x + threadIdx.x;
03      double calc = x[i]*w[i];
04      __shared__ double shared_net;
05
06       __syncthreads();
07      shared_net += calc;
08       __syncthreads();
09
10     net[0]=shared_net;
11  }
12
13   inputs=32;
14   double net[1] = {0.0};
15   double *cuda_x, *cuda_w, *cuda_net;
16   int nrbytes = (inputs * sizeof(double)) ;
17
18   cudaMalloc((void **) &cuda_x, nrbytes );
19   cudaMalloc((void **) &cuda_w, nrbytes );
20   cudaMalloc((void **) &cuda_net, sizeof(double) );
21
22   cudaMemcpy(cuda_x, x, nrbytes, cudaMemcpyHostToDevice);
23   cudaMemcpy(cuda_w, w, nrbytes, cudaMemcpyHostToDevice);
24   cudaMemcpy(cuda_w, w, sizeof(double), cudaMemcpyHostToDevice);
25
26   dim3 dimGrid (1);
27   dim3 dimBlock (inputs);
28   propagation<<<dimGrid,dimBlock>>>(cuda_x,cuda_w,cuda_net);
29
30   cudaThreadSynchronize();
31   cudaMemcpy( net, cuda_net, sizeof(double), cudaMemcpyDeviceToHost );
```

Figure 7. Implementation in CUDA

The kernel implementation, which will be executed at the same time by the 32 block threads, is found in the beginning of the code (lines 01 to 11). In this implementation the variable "calc" receives the result of the thread processing (line 03). As this variable is in the local memory, just the thread which instantiated it can access it. After, this result is added to the "shared_net" variable value (lines 6 to 8). As it had been qualified as "__shared__", it is located in the shared memory (all block threads have access), and it is necessary to create a barrier (lines 6 and 8). Finally, when all threads finish (line 30), the propagation result is passed to the host (line 31).

Despite of the simple example, it can show that the most complex part of the calculation (line 03) was totally executed in parallel by the GPU. The expansion, for example, to an NNA with 16 neurons in its hidden layer, where the propagation of this layer can be made in parallel by a 16-block kernel, can be done easily.

5. Conclusions e Further Work

Computer games are a special way to work with different abilities of the humanbeing, because different skills, as memory, attention, creativity and reasoning, which could be tested and analyzed. The use of AI in games still is very restrict, because the main games focus is the graphical interface (engines, renderization, etc), but it could bring more realist action, with a level of uncertainty into the game.

Develop a tool to make easier the insertion of AI techniques in computer games could be the way to a massive use of them. In this paper, we presented the conceptual propuser of this framework by UML class diagrams. We also did a requirements analysis of two AI techniques, where we defined the initial variables and steps of execute them. Finally, we presented the CUDA architecture, that will be used in our framework to parallel execution of AI techniques.

The next step is the implementation of these modules in Java language. The validation of the framework will be done in some small computer games, included players and environment actions to evaluate their performs with unpredictability.

6. Acknowledgments

Diana F. Adamatti is financed by FAPERGS (Brazil), grant number 0902806/2009.

References

[1] Caglayan, A., Harrison, C. (1997).Agent SourceBook - A complete Guide to Desktop, Internet and Intranet Agents. New York: Wiley Computer Publishing, 349 p.

[2] Funge, J. D. (2004). Artificial Intelligence for Computer Games. A K Peters, Wellesley, Massachusstes, USA.

[3] Genesereth, M.Love, N., Pell, B. (2005). General game playing: Overview of the AAAI competition. AI Magazine .

[4] Laird, J. E., Lent, M.(2001). Human-level ai's killer application: Interactive computer games. AI Magazine.

[5] Mobascher, B., Burke, R., Bhaumik, R., Sandvig, J. J. (2007). Attacks and Remedies in Collaborative Recommendation. IEEE Intelligent Systems.

[6] Nareyek, A. (2001). Constraint-Based Agents - An Architecture for Constraint-Based Modeling and Local-Search-Based Reasoning for Planning and Scheduling in Open and Dynamic Worlds. Berlin: Springer.

[7] Resnick, P., Varian, H. R. (2001). Recommender Systems. Communications of the ACM, New York.

[8] Ricci, F., Nuygen, Q.N. (2007). Acquiring and Revising Preferences in Critique-Based Mobile Recommender Systems. IEEE Intelligent Systems.

[9] Whitley, D. (2009). A Genetic Algorithm Tutorial. Technical Report: Colorado State University. 2002. Available in http://xpusp.sourceforge.net/ga_tutorial.ps.

[10] Russel, S., Norvig, P. (2009). Artificial Intelligence: A Modern Approach.

[11] Nguyen, H. T., Walker, E. A. (2006). A First Course in Fuzzy Logic.

[12] Haykin, S. (1994). Neural Networks: A Comprehensive Foundation. Prentice Hall: USA.

[13] Sommerville, I. (2004). Software Engineering. Addison Wesley: USA.

[14] NVIDIA, NVIDIA CUDA C Programming Guide Version 3.2. NVIDIA Press: USA(2008).

[15] NVIDIA, CUDA Technical Training Volume I: Introduction to CUDA Programming . NVIDIA Press: USA, (2008).