SCORE: a proposal to design Cognitive Agents' behavior on Interactive Computer Games

Leonardo S. Cunha
PUCRS, Programa de Pós-Graduação em Ciência da Computação,
Porto Alegre, RS, Brazil, 90610-900
sewald@inf.pucrs.br

and

Lucia M. M. Giraffa
PUCRS, Programa de Pós-Graduação em Ciência da Computação,
Porto Alegre, RS, Brazil, 90610-900
sewald@inf.pucrs.br

Abstract

The use of computer games as testbeds for research projects in the Artificial Intelligence (AI) field can be considered as some kind of tradition. Some classic board and card games such as Go, Chess and Checkers were and are extensively used. However, AI researchers are beginning to explore the use of real-time, commercially-exploited interactive computer games in their research, making this a very interesting research field to experiment, and to explore the limits and the possibilities of some AI techniques. Interactive computer games can provide the robust environments researchers need to focus on intelligent agent research. This paper presents the research work that has been developed in our group: the SCORE project (Simulator for COgnitive agent's behavioR), where we use a Belief-Desire-Intention (BDI) agent specification formalism (associated to X-BDI and E-BDI tools) applied to an interactive computer game environment called *Unreal Tournament* (UT). This paper also presents some aspects regarding agent technology applied to interactive computer games. We emphasize in the application of cognitive agent modeling applied to game characters. The focus is not only to contribute for the game industry itself, but mainly to evaluate the potential of these technologies to solve problems related to user modeling, specially in applications where user's behavior modeling is needed.

Keywords: Artificial Intelligence, Cognitive Agents, Simulation and Computer Games.

1 Introduction

Interactive computer games have been showing many innovations concerning about performance, interfaces and project techniques. According to Battaiola [1], the starting point was the 70's with the game named *Pong* (www.gamesdomain.co.uk/faqdir/pong.txt). It was the first gaming market hit where two players would compete in a tennis simulation. Since *Pong*, interactive entertainment applications have shown a constant evolution, turning into repositories of technological innovations in different fields, such as Computer Graphics (CG), with real three-dimensional (3D) graphics and total freedom of movement, and Artificial Intelligence (AI), making use of different techniques, in particular, agent technology [2].

The first 16 bit computers appeared in the 80's, like the IBM Personal Computer (PC), Atari ST and the AMIGA 500, and they caused a significant improvement in computer games development. Some 3D simulation games also appeared in the same period, like *Flight Simulator* (http://www.crosswinds.net/~andysflightsim). However, one game considered a milestone of computer games' technological evolution appeared in the 90's: *Quake* (http://www.quakeworld.com), developed by ID Software. It was the first real 3D game for PC's (until then, developers would use programming "tricks" to create fake 3D environments (3D illusion), as happened in *Doom* (http://www.idsoftware.com)).

The focus of computer games' evolution was, until then, concentrated in CG. In 1998, *Unreal* appears (http://unreal.epicgames.com). A Quake-style game, released by Epic Megagames. It showed a significant progress, not only in the applications of CG techniques, but mostly in the application of AI techniques. Rule-based AI approaches were used in order to improve the game's design and playability. The techniques used were Finite State Machines (FSM) and Fuzzy State Machines (FuSM), aside of the implementation of a programming technique called *Extensible AI*, which allows the player to add or modify the game's AI complexity, extending the game's features (these techniques will be described in the next section). Right after *Unreal*'s release, Epic developed *Unreal Tournament* (UT), a game using the same architecture used in *Unreal*, but with a stronger multiplayer approach.

Today's modern computer games have state-of-the-art 3D graphics, many multiplayer features, allowing players to compete against each other or play cooperatively over a network and fairly complex AI implementations. But more importantly, modern computer games are being developed with the use of architectures that allow third-party developers, users and researchers to expand, add or modify game content. *Unreal*, UT and games created using the *Unreal* architecture (which we will describe later on this paper) all fit into this category.

According to Laird [3], real-time interactive computer games offer robust environments for researchers to test and develop AI techniques, aside of the fact that games are relatively cheap and accessible, comparing to industrial or commercial applications. These games have environments populated with human and/or characters controlled by computers. Since a direct addressing between interactive game characters and agents can be done, as stated by Laird and others in Cunha [2], these games become a rich laboratory for AI research, especially in what concerns agent development. Moreover, interactive computer games avoid many of the criticisms leveled against research based on simulations: because researchers do not develop them, the games avoid embodying preconceived notions about which aspects of the world designers can simulate with ease and which aspects are more difficult to simulate. These games constitute real products that create real environments with which millions of humans can interact vigorously. For these attributes, the use of real-time interactive computer game environments in intelligent agent research is an interesting field to experiment and explore.

Having this thought in mind, the GAMES AND AI (JEIA) research group at PPGCC/PUCRS is developing SCORE, a research project which aims to integrate an agent specification formalism, programming language and tool called X-BDI with the game UT. Both applications and their features will be described later on this paper.

This paper presents a description of our research work (section 4). Section 2 presents a description of the main AI and programming techniques used in current computer games, focusing in the agent technology (section 3). Final considerations and results achieved with the use of agent technology are presented in section 5. References are presented at the end of this paper.

2 AI and AI-related programming techniques currently being used in interactive computer games

We developed a study (survey) about the set of techniques, which represent the state-of-the-art, in what concerns the use of AI and AI-related programming techniques applied to computer games. The main techniques are [2]:

- *Finite State Machines* (FSM's): rule-based approaches such as FSM's and *Fuzzy State Machines* (FuSM's) are the most used AI techniques in computer games, because they are easy to implement and for the fact

that they are consolidated as AI tools in the computer game industry. A FSM is composed by a set of states (including a starting state), a set of inputs, a set of outputs and a state transition function. This function computes the entries and the current state and outputs an unique new state and a set of outputs. FSM's are usually represented through state transition diagrams, and can be easily implemented through nested *switch-case* commands. For more complex modeling, the FSM's can be built under a graph hierarchy, so that each node in a certain hierarchy level can be expanded to reveal it's dependent hierarchy, and so on, until the last level, expanding a FSM. These are the *Hierarchical Finite State Machines* (HFSM's), and they provide an efficient way for FSM modular modeling. According to Kantrovitz [4], the FuSM's are based on Markov chains. Weights are associated to states and transition functions and rules are established to compute the weights of the future states. An example of a game which uses FuSM's is *Call to Power* (http://www.calltopower.com), a strategy game where the player controls civilizations whose profile and characteristical traits were modeled with the use of FuSM's. The use of FSM's allow the user to build elements with a relatively complex behavior, as showed in the game *Unreal* (http://www.unreal.com), where the enemies would "take decisions" based on the player's actions;

- *Extensible AI*: this is not exactly an AI technique, it's more likely to be considered as a programming technique. Through this programming style, the player has the possibility to create character behaviors or modify pre-existent character behavior. *Extensible AI* usually appears as in the form of *script* languages (which can look like C or C++). The game developers must build a *script* compiler which is inserted in the game's executable file, or as a separate program shipped with the game). The *scripts* are based on function calls to the game's internal AI subsystem, which cannot be modified by a regular *Extensible AI* implementation. According to Woodcock [5], many games shipped in the last few years implement this technique. It is a trend that started with games such as *Duke Nuke'em 3D* (http://www.dukeworld.com) and *Quake*;

- Search algorithms – the *A-Star* (A*): According to Woodcock [6], the A* is the most used search algorithm in computer game design and implementation. The A* uses a heuristic function which determines the quality of each of the possible states (nodes). The function estimates the cost of the paths towards the destination, passing through the current node, and it chooses the best way to achieve it. The quality of the node is measured by lowest cost among all candidate nodes. The performance of its implementation depends on the heuristics enclosed in the heuristic function. A bad heuristic function can drastically reduce the algorithm's speed, or produce incorrect routes. According to Patel [7], in order to achieve the best results, the heuristic function must be admissible. It means that it must be an underestimate of the actual cost of moving from the current node to the goal node. Usually, game developers are very familiar with search algorithms. Thus, now they focus on the association of search techniques with specific situations, such as in pathfinding associated with *terrain analysis*[1], a situation usually found in strategy games such as *Age of Empires II: The Age of Kings* (http://www.ensemblestudios.com/aoeii/index.shtml), a game where the player must develop ancient civilizations;

- Neuronal Networks and Genetic Algorithms: According to Hayking [8], Neuronal Networks (NN's) use a massive interconnection of computational cells called "neurons" or "processing units". The links between these neurons are called synapses. Each neuron has an associated value (weight), which is used to store acquired knowledge. The neuronal net's algorithm can learn and keep the changes of these weights in order to improve the neuronal network. A neuronal net's learning process is called supervised if the expected output is known. Otherwise the learning process is called unsupervised. Genetic algorithms (GA's) are computational models inspired on human evolution. They typically represent knowledge through binary or Boolean attributes. According to Whitley [9], a regular GA implementation starts with a set population with some attributes (chromosomes). These structures are evaluated by a fitness function and reproduction opportunities are offered. The set of attributes, which represents a better solution to the problem, has a better chance to reproduce. This evolution process can be associated to the learning process for GA's. There are some projects which explored a combination of GA's and NN's, and one of the most significant is the *NeuralBot* (http://www.botepidemic.com/neuralbot), a computer-controlled opponent (*bot*) created to be used in the game *Quake II* (http://www.quakeworld.com). The *bot* uses a neuronal net to control its actions and a GA to train its neural net. By this way the *bot* does not need any sort of pre-programmed behavior: it can learn and adapt itself using the environment inputs;

- Artificial Life (A-Life): according to Woodcock [6], the *A-Life* representation techniques provide flexible

[1] The *terrain analysis* is a programming technique which is used to point out locations on the game maps which can be difficult for the game elements to move through, such as bridges or mountain passes. A good *terrain analysis* can ouput valuable information for the game's pathfinding system so that the latter can solve more complex search prolems.

ways to create realistic behavior in game characters. *A-Life* aims to simulate the behavior of real world living organisms through different methods, such as rule-based approaches, GA, among others. Instead of implementing a variety of complex behaviors, the *A-Life* approach divides these complex behaviors in small parts, in order to simulate simpler behaviors. A decision-taking hierarchy, used by the game elements to decide what actions must be taken interconnects these small parts. Combinations of low-level behavior sequences can automatically generate higher level behaviors, without the necessity of implementing complex behaviors. *A-Life* techniques are usually used in simulators. However, some action and strategy games, such as *Unreal* (http://www.unreal.com) and *Age of Empires* (http://www.ensemblestudios.com/aoe), already use *A-Life* approaches to control group movement (*flocking algorithms*);

- *Software Development Kits* (SDK's): SDK's, or simply *toolkits*, implement different AI techniques ready to be used by game developers. For this attribute SDK's also aren't AI techniques, but can be considered as AI-related programming techniques. They can speed up the application development process, since developers just have to use the functions implemented in the toolkit, instead of implementing their functionality (for example, when using a FSM approach. The developers can use a ready-made toolkit implementation of general FSM functionality and tune it up for their project). Some SDK's were developed to be used specifically with game projects, while others have a wider usage scope. Some SDK's are: *Motivate* (http://www.motion-factory.com), *Spark!* (http://www.louderthanabomb.com) e *DirectIA* (http://www.animaths.com).

The techniques shown above are part of the set of techniques which is currently being widely used by the game development community. In our research work, we are using a game which implements FSM's in its AI subsystem. This AI technique interacts with our agents' specification trough a specific formalism, as we will explain forward on sections 3 and 4.

3 Computer games implemented with agent technology

According to Nwana [10], the term "agent" is widely used in different research areas, but there is no consensus concerning its definition. According to Wooldridge [11], an agent is a computer system that is capable of independent action on behalf of its user or owner. A Multiagent system is one that consists of a number of agents, which interact with one another. Multiagent systems provide a new tool for simulating societies, which may help shed some light on the various kinds of social processes. Thus, agents can be considered as a tool for understanding human societies, or other societies based on the human ones. In order to successfully interact, agents must have the ability to cooperate, coordinate, and negotiate. According to Nwana [10], Russel [12], and Wooldridge [11] the agents have different characteristic attributes (properties). However, some properties are necessary to be observed when building an agent:

- Autonomy: refers to the fact that agents can operate in an independent way, without human supervision. Agents usually implement this property, but in a certain degree;
- Reactivity: refers to the agent's ability to react to environment inputs. A reactive system is one that maintains an ongoing interaction with its environment, and responds to changes that occur in it (in time for the response to be useful);
- Social ability (interaction): refers to the interaction with other agents (and possibly humans) via some kind of agent-communication language.

We adopted the definition used by Nareyek [13], which defines an agent as an entity, which has goals, that has the capacity of perceiving certain properties in the environment. They can sense the surrounds and can act in this environment, and some of these actions and/or perceptions can be done through communicating with other agents. According to many authors mentioned by Vicari and Giraffa [14], there is a taxonomy being widely used in the Distributed Artificial Intelligence (DAI) community, which classifies agents in two groups: reactive and cognitive agents. The differences between them are centered on the fact that reactive agents consider only current environment information as inputs for its decision process. The cognitive agents are deliberative, and some of them can learn with their experiences. It means that they can plan and execute their own created plans. However, the modeling and implementation of reactive agents should not be considered a simple task. We can find fairly sophisticated reactive agent implementations in computer games, for example. This taxonomy has more didactical application than a real/practical applicability. By observing a Multiagent System (MAS) it is difficult to classify the agents as cognitive or reactive. It is an internal project issue, which sometimes is not explicit in the project's description.

According to our previous research [2], the cognitive agent architecture is typically unable to act fast and adequately in unpredicted situations, while reactive agents are unable to discover alternatives to their behavior when the

environment status is too different from their initial goals, making them less flexible than cognitive agents. Table 1 presents some comparisons between reactive and cognitive agents in what concerns their basic features [15].

Table 1. Reactive Agents X Cognitive Agents

Reactive Agents (non-deliberative)	Cognitive Agents (deliberative)
Reactive agents do not have an internal symbolic environment representation	They have goals and an internal world representation, which contains information about application requirements and action consequences that can be explicitly represented
Action selection is directly associated with the occurrence of a set of events in the environment	Action selection is done through an explicit deliberation over different options. For example, by using an internal world representation, a plan or even a function which evaluates an action according to its use
Architectures have a fixed, rule-based action control mechanism	They have an internal planning system based on successive refinements, using the information provided by the world representation to build a plan which can reach the agent's goals
Demonstrate excellent performance in real-time environments, although there is the need of a previous effort to determine specific solutions to the possible situations	Lack of speed in real time environments. The amount of time needed to analyze the situations, to build and to reuse plans, typically can make the agent very slow agent to act in a real time environment
In general they don't have explicit goals which can be arbitrated and altered during the execution of the system	Given a certain goal and the knowledge that a certain action will lead it to this particular goal, then the agent selects this action
Built with the use of simple control mechanisms such as FSM's or rule-based approaches	Originated from the classic planning systems, which output a sequence of correct actions (plans) to reach a certain objective

An example of a game which uses reactive agents is *Guimo* (http://www.jackbox.com/guimo), a game released in 1997 by a Brazilian company named Jack in the Box Computing. The agents were built using the HFSM approach. The HFSM has states such as *Wander*, *Fallback* and *Attack*, and each one of these "macro-states " has an internal FSM, which executes the specified actions.

Nowadays, few computer games incorporate cognitive agent technologies, as for instance *Black and White* (http://www.lionhead.com/unshocked/bw), a game released by LionHead in 2001. The agents were modeled based on a *Belief-Desire-Intention* (BDI) architecture. The agent receives an "inventory" of locations of certain points. Of those descriptions, the agent then creates "opinions" over which types of objects are more appropriate to satisfy its goals.

In order to solve the main restrictions of these two architectures, the hybrid architectures were adopted. A hybrid architecture combines reactive and cognitive techniques. According to Nareyek [13], hybrid agents use an off-line planning system (a planning system which is not executed in real time), for the generation of plans in a higher abstraction level, while decisions are made on the lower level. Refinement alternatives for plan steps are handled by reactive systems.

An example of a hybrid agent application is the *Multi-Cooperative Environment* - MCOE (http://www.inf.pucrs.br/~giraffa/mcoe/mcoe.html), a multimedia educational game, created as a prototype for Giraffa's [16] PhD thesis. The thesis work proposed to model an Intelligent Tutoring System (ITS) through the use of agent technology using a Multiagent System (MAS) architecture. The ITS was conceived as a hybrid MAS composed of a reactive environment and a cognitive core, which interact in order to extend the students' quantitative and qualitative information, this way allowing the tutor to select more appropriate teaching strategies to a certain student profile.

4 Integrating Computer Games and Intelligent Agent Research

Some modern computer games have their *engines*[2] implemented as *Dynamic Link Libraries* (DLL's). They are files, which store the system's object/function libraries. These DLL's are used to manipulate game data (textures, graphical elements) and interact with certain part of the system which typically are external to the engine itself, such as music/sound effects, AI and other subsystems.

Through the use of some tools (in most cases, shipped with the game) these subsystems' source code can be viewed and manipulated, allowing the user to change many game attributes without the necessity of acquiring the game engine's source code. These subsystems can be considered as the engine's programmable interfaces (since the game engine interprets the code for these subsystems). They allow access to internal game functions through programming techniques such as *Extensible AI*. Most of these interfaces are accessible with the use of script languages, whose commands and reserved words assigned to game AI tasks are usually associated with low-level actions. The developer and/or researcher could directly program a character's behavior by using these interfaces. One game that fits into this category is UT.

The SCORE project creates/uses the game UT as a testbed. UT was created by the company *Epic Megagames* and released in 1998. The game belongs to the *first-person shooter* (FPS) game category, where the player visualizes the environment from the game character's perspective.

The game's architecture is quite complex, considering a computer game project. The approach consists of the use of a Virtual Machine, a Compiler and script and byte code (similar to the Java language architecture), as shown in Figure 1.

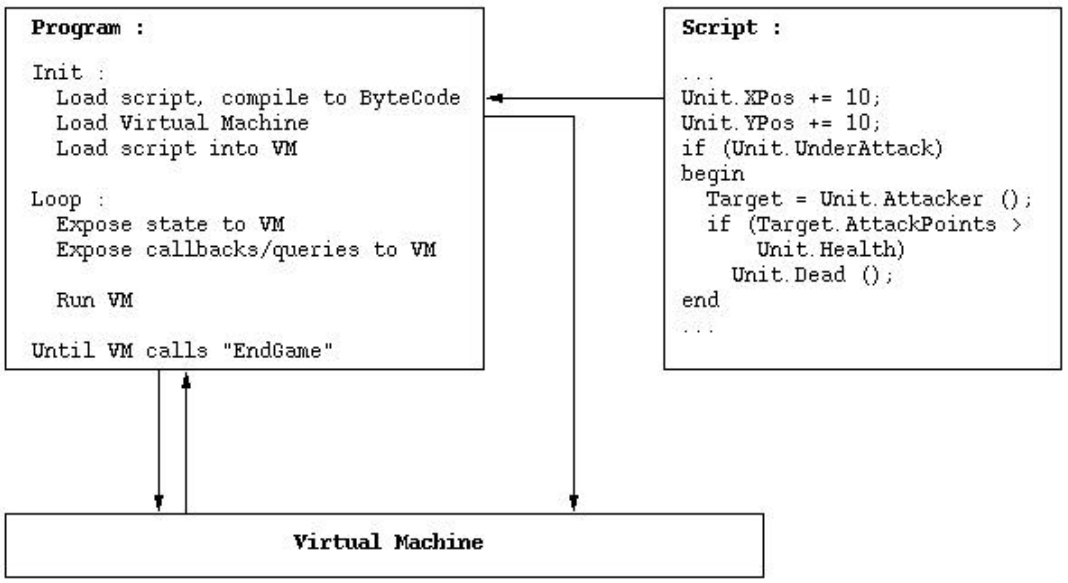

Figure 1. UT architecture scheme

The *Init* sequence compiles the script code into byte code and loads it on the Virtual Machine to be executed. From this point, the game's execution loop begins. During the game's execution, the system's state changes. The Virtual Machine executes functions and service calls until an "EndGame" request is called [17].

The *Script* part of the scheme is represented by UT's built-in, integrated script language, called *UnrealScript*. The script language was created to provide a personalized game programming language, which would provide native support for features such as states, time and network communication. UT's development team first used *UnrealScript* during the game's implementation phase. Now many unreal players who have programming knowledge use the script language.

The language is object-oriented, mixing C++ and Java features. The implemented code is compiled by using the UCC tool, a compiler created by UT's development team to be used with the language, and then be executed by the *Unreal Virtual Machine*, which is basically represented by the game's engine.

[2] In what concerns computer game development, an *engine* consists of a set of methods/functions and data structures created to ease the manipulation of game data, thus assisting programmers in creating computer games. The engine is tipically considered as a game's core.

The script language is part of a powerful editing tool called *UnrealEd*, and can also be accessed through it. UT's development team also first used the editor to build the game, and now this editor is being shipped with the game itself, accessible to all users who wish to change/manipulate the game elements. Figure 2 shows a screenshot of *UnrealEd*.

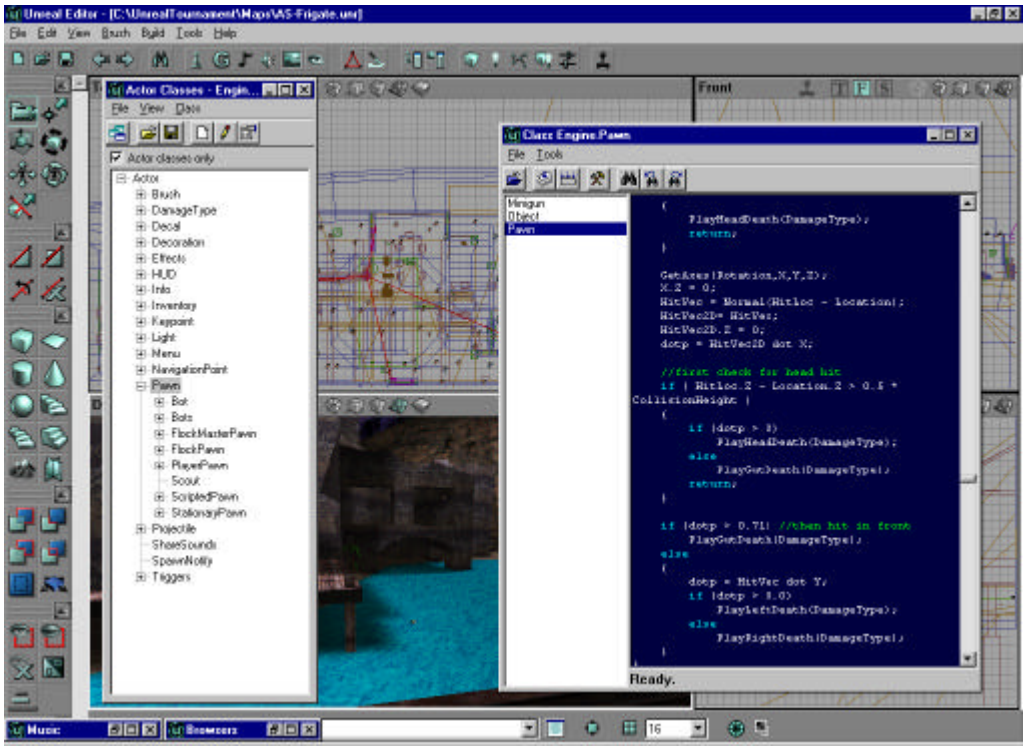

Figure 2. A screenshot of *UnrealEd's* Integrated Development Environment (IDE)

All the game elements (music, sound effects, textures, characters, and weapons, enemy AI) can be manipulated through the editor and/or *UnrealScript*. The screenshot above shows two script-related windows: the one on the left shows a part of the game's object hierarchy, while the one on the right shows the *UnrealScript* code for the class *pawn*, which represents a game element which has AI capabilities. The user can change the code or inherit the properties of other superclasses and then recompile the game's script code[3], this way changing the game.

Unreal and *UnrealScript* were chosen to be used in this project because there is available documentation, aside of our interaction with the development team members.

Through the use of the *UnrealScript* (which can be considered as UT's programmable interface), it would be possible to manipulate character behavior. However, it would be quite hard task to create complex (intelligent) cognitive agent behavior by direct script language manipulation. Fortunately, UT's game interface can be controlled via external software (thus controlling character behavior).

At this point, there is the necessity to use tools in order to reduce agent modelling and implementation complexity. Our research group has two available tools to attend this issue:

- X-BDI: X-BDI is an agent development and testing tool based on the now common concept of beliefs, desires and intentions [18]. Derived from a formal model of BDI agents [19], X-BDI implements the algorithms that deal with the interaction of mental states, namely: how to keep beliefs consistent, how to keep intentions consistent (internally, and with beliefs), how to derive intentions from desires and plans from intentions. Therefore, X-BDI provides a tool that, when fed with the description of an agent in terms of its beliefs and desires and given input from the environment, is capable to manage these mental states and produce sequences of actions which satisfy the agent's desires, and passes them to the environment. X-BDI is not seen as a complete agent but as a cognitive kernel that is to be part of an agent.

- E-BDI: the E-BDI system is a BDI agent programming editor based on the X-BDI environment, making it

[3] It is important to state that by using the editor, the user cannot change the game's engine. In fact, the editor uses the engine's resources to allow the user to manipulate the game elements.

more operational and usable. The editor allows the developer to define the set of BDI mental states and their interrelations by using both a text and graphic-oriented interface [20]. In this way, the E-BDI editor addresses one of the main issues in BDI agent programming the complexity to visualize and debug large BDI mental state descriptions. The developer can analyze the defined mental states, making it easier to detect the presence contradictions or deadlocks [20].

Our research project is to use UT as an environment for intelligent agent research using the X-BDI environment to control character behavior, while programming and defining the mental states using E-BDI. We are using some previous results reached by Mora [19], Giraffa [16], Vicari [14], Zamberlam [20] and Goulart [21] to control character behavior in interactive computer games. We are using the mental state approach (modeling with the E-BDI editor), in a similar way as Laird's project [3], where the Soar[4] architecture is used.

The X-BDI system has a higher abstraction level than the game's built-in *script* language, making it an essential tool for cognitive agent modelling. Thus, it is necessary to build an intermediate layer to establish the communication between the game's programmable interface and the agent's cognitive kernel, as depicted in Figure 3 below.

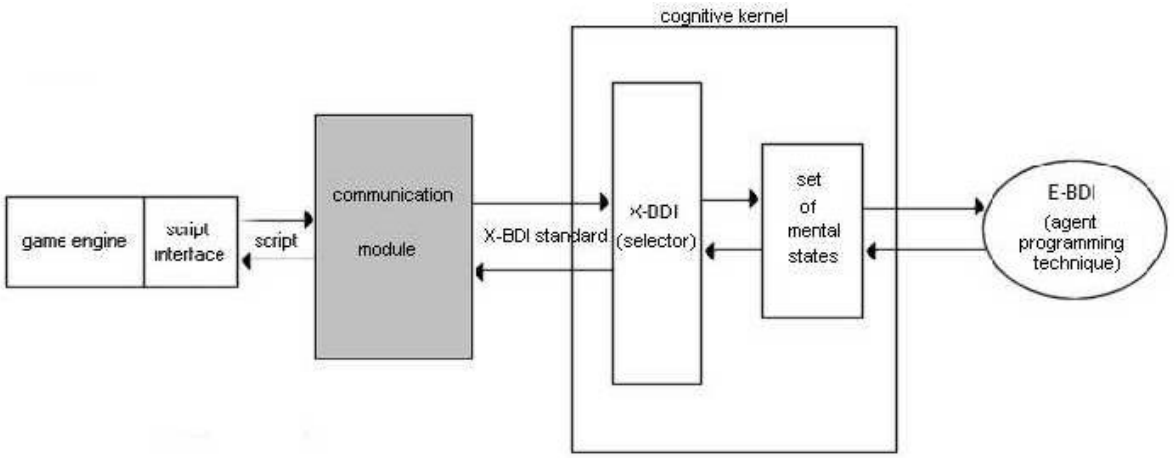

Figure 3. controlling computer game characters of an interactive computer game via X-BDI

In order to establish such a communication level, a two-way grammar was defined. In the game -> X-BDI way, the grammar maps game outputs, coded in *UnrealScript*, as inputs representing the agent's environment sensoring, coded in the BDI format used by X-BDI. This information is then processed by X-BDI. The vice-versa process, translating X-BDI-generated plans into sequences of actions, coded in *UnrealScript*, to be executed by the game characters, is also made using this grammar.

Code samples from *Unreal's script* code and X-BDI agent description file are shown on Table 2.

Table 2. code examples from *Unreal* script and X-BDI

Unreal script code	X-BDI code

`Else if (Orders == 'Patroling')`
`GotoState('Patroling');`
`else if (Orders == 'Attacking')`
`GotoState('Attacking');`
`else if ( Orders == 'Ambushing' )`
`GotoState('Ambushing','FindAmbushSpot');`
`else if ( (LikelyState != '') && (FRand() < 0.35) )`
	`Identity(rbt).`
	`Des(rbt,loaded_battery,0.5).`
	`Des(rbt,safe(O),0.3) if`
	`bel(rbt,entrance(O)).`
	`Act(rbt,load) causes`
	`bel(rbt,loaded_battery).`
	`Bel(rbt,-loaded_battery) if`

[4] Among the projects where the Soar architecture was used, let us highlight the research project which served as a motivation for our work: the *human-level AI* research project, coordenated by Laird [3], where the Soar architecture was integrated with the game *Quake II*.

```
                GotoState(LikelyState, LikelyLabel);
        Else
         StartRoaming();
...
state Attacking{
ignores SeePlayer, HearNoise, Bump, HitWall;
        function ChooseAttackMode()
        {
                local eAttitude AttitudeToEnemy;
                local float Aggression;
                local pawn changeEn;
                local TeamGamePlus TG;
                local bool bWillHunt;
                bWillHunt = bMustHunt;
                bMustHunt = false;
                if ((Enemy == None)||(Enemy.Health <= 0))
                {
                        WhatToDoNext('','');
                        return;
                }
```

```
   bel(rbt,charged_ok,T).
act(rbt,keep(0)) causes
   bel(rbt,safe(0)).
Act(X,entrance(0)) causes
   bel(rbt,enter(0)).
Act(rbt,keep(0)) causes
   bel(rbt,-enter(0)).
```

As shown in Table 1, the X-BDI language is similar to PROLOG, having low complexity in what concerns it's syntax. The *UnrealScript* code excerpt shown above depicts the native implementation of states, used mainly for AI purposes, reducing the complexity to program character behavior (in regular programming languages, programmers would have to use C/C++ "switch" statements to represent states). The state concept implemented in *UnrealScript* defines sequences of actions, which can be associated to the actions performed by a X-BDI agent. The conditions that determine if an agent will or will not enter a given state are associated with beliefs of a X-BDI agent.

Our project intends to make possible to model and program high-level behavior for BDI agents, and visualize the results in an interactive game environment. This way, computer game environments work as good testbeds for intelligent agent research because they allow quick feedback for user and designers.

5 Final Considerations

The AI research field has developed many works using classic games such as chess, checkers or puzzles, and so on. However, modern computer games are, in their majority, played in real time, in virtual environments, which involve a high level of dynamism and interactivity. Thus, real time computer games are an interesting topic for study and implementation of AI techniques. There is also the fact that there are very few research projects involving modern computer games in the academic community. There is some kind of misunderstanding or resistance over the use of interactive computer games as powerful testbeds for AI techniques. The works Laird [3] and Adobbati [22] have developed with the use of real-time computer games are good examples about the potential of such applications. Considering that the first Brazilian Workshop on Games and Digital Entertainment will be held on October, 9th, 2002, in Fortaleza-Brazil, this shows the importance and the recognition of interactive computer games as a research area.

In fact, the lack of scientific material on the subject is one of the main difficulties in the elaboration of our research work. According to Laird [17], the main reason that makes the academic community ignore real-time interactive computer games is associated to the fact that, usually, the objective of these game's AI systems is not to create intelligent characters, but to improve the game's playability level through the illusion of intelligent behavior. Another reason to justify the shortage of scientific material on the subject would be the fact that many game developers like to "reinvent the wheel", creating their own game development methodologies instead of using ready-made ones. Sometimes game programmers code their systems in an extremely complex and disorganized way, making it difficult to understand, however working in an optimized way. The JEIA research group from PUCRS is trying to contribute to modify this unfavorable scenario.

As examples of the research being developed by the group, we can cite: MCOE [16]; RL_MCOE, proposed by

Callegari [24], is an application of reinforcement learning techniques using the work implemented by Giraffa and Vicari [16; 14]; TCHE by Mazzorani [25], and QUERO-QUERO by Comunello [26] are educational games created to assist children to develop basic math concepts. These game's environments were developed by multidisciplinar teams and tested in real classroom environments. The JEIA group has been developing research involving agent technology applied to educational game modelling and implementation. The group is now starting to develop applications involving real-time interactive computer games. More detailed information is available at http://www.inf.pucrs.br/~giraffa/jogosIA.

Our work described in this paper also intends to demonstrate the potentiality of BDI agent techniques for game projects. As important contributions from this work, we can mention:

- Provide a standard for X-BDI output manipulation: by having a standard way to manipulate X-BDI output, we can expand the communication layer in order to enclose different languages, enlarging it's usage scope;
- Contribute to spread out interactive computer game research in the academic research community;
- Provide tools to develop an application which will allow behavior modelling and programming through E-BDI's associated programming technique;
- Contribute with BDI agent research.

As future work we intend to test the prototype with players and observe the degree of difficulties/problems concerning our grammar syntax, user's interface interaction, debugging library to create a better help system, and so on.

Acknowledgements

DELL computers support this project.

References

[1] BATTAIOLA, A. L. **Jogos por Computador – Histórico, Relevância Tecnológica e Mercadológica, Tendências e Técnicas de Implementação.** XIX Jornada de Atualização em Informática, 2000, ed. Universitária Champagnat, pp. 83-122.

[2] CUNHA, L. S.; GIRAFFA, L. M. M. **Um estudo sobre o uso de Agentes em Jogos Computadorizados Interativos.** PPGCC/PUCRS, Porto Alegre, 2001. Technical Report (available at http://www.inf.pucrs.br/ppgcc/).

[3] LAIRD, J. E. **Using a Computer Game to Develop Advanced AI.** Computer, 34, (7), 2001.

[4] KANTROVITZ, M. **Internet FAQ Consortium**. Captured in April 2001. *Online*. Available on the Internet at: http://www.faqs.org

[5] WOODCOCK, S. **Game AI: The State of the Industry**. Captured on March 2001. *Online*. Available on the Internet at: http://www.gamasutra.com/features/19990820/game_ai_01.htm

[6] WOODCOCK, S. **Game AI: The State of the Industry**. Captured on April 2001. *Online*. Available on the Internet at: http://www.gamasutra.com/features/20001101/woodcock_01.htm

[7] PATEL, A. J. **Amit's thoughts on pathfinding**. Captured on April 2001. *Online*. Available on the Internet at:: http://theory.stanford.edu/~amitp/GameProgramming/

[8] HAYKING, Simon. **Redes Neurais: Princípios e Prática**. Bookman, 2001.

[9] WHITLEY, Darrel. **A Genetic Algorithm Tutorial**. Statistics & Computing. (4), ,1994.

[10] NWANA, Hyacinth S. **Software Agents: An Overview**. Knowledge Engineering Review. (11), 3, 1994.

[11] WOOLDRIDGE,M. **An Introduction to Multiagent Systems.** England: John Willey & Sons, 2002.

[12] RUSSEL, S.; NORVIG, P.; **Artificial Intelligence: A modern Approach**. Prentice-Hall, 1996.

[13] NAREYEK, Alexander. **Intelligent Agents for Computer Games**. Captured on April 2001. *Online*. Available on the Internet at: http://www.ai-center.com/references/nareyek-00-gameagents.html

[14] VICARI, R.M; GIRAFFA, L.M.M. **The Use of Multi Agent Systems to Build Intelligent Tutoring Systems** In: International Journal of Computing Anticipatory Systems. The American Institute of Physics (AIP), 2002.

[15] DÁMICO, C.; GIRAFFA, L.; BERCHT, M.; LADEIRA, M. **Inteligência Artificial: uma abordagem de agentes**. CPGCC/UFRGS, 1995. Research Report N.257

[16] GIRAFFA, L.M.M.; VICCARI, R.M. **Estratégias de Ensino em Sistemas Tutores Inteligentes modelados através da tecnologia de agentes**. Revista de IE/SBC. (6), 2, 1999. (Brazilian Journal of Computer Science applied to Education)

[17] CUNHA, L. S.; GIRAFFA, L. M. M. **UnrealScript Language Syntax.** PPGCC/PUCRS, Porto Alegre, 2001. Technical Report (available at http://www.inf.pucrs.br/ppgcc/).

[18] WOOLDRIDGE, M. **Reasoning about Rational Agents.** Massachusetts: The MIT Press, 2000.

[19] MÓRA, M.C.; LOPES, J.G.; COELHO, J.G.; VICARI, R. **BDI models and systems: Reducing the gap.** In: Agents Theory, Architecture and Languages Workshop. Lecture Notes on Artificial Intelligence, Springer-Verlag, 1998.

[20] ZAMBERLAM, A.O.;GIRAFFA, L.M.M.; MÓRA, M.C. **E-BDI: um editor para programação orientada a agentes BDI.** In: III ENIA Encontro Nacional de Inteligência Artificial, 2001, Fortaleza. XXI Congresso da Sociedade Brasileira de Computação. Fortaleza. Proceedings: SBC, 2001

[21] GOULART, R. R. V.; GIRAFFA, L.M.M.; MÓRA, M.C.; **Auxiliando o tutor na gerência das informações do ambiente e dos alunos.** XI SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO, 2000, Maceió, Alagoas. Proceedings: SBC, 2000.

[22] ADOBBATI, R. et al. **GameBots: A 3D Virtual World Test-bed for Multi-Agent Research.** Proceedings of the 2nd. Workshop on Infrastructure for Agents, MAS and Scalable MAS, ACM Press, New York, 2001, pp. 47-52.

[23] LAIRD, J. E. **Game AI: The State of the Industry, part two**. Captured on March 2001. *Online*. Available on the Internet at: http://www.gamasutra.com/features/20001108/laird_01.htm

[24] CALLEGARI, D. A. **Aplicando aprendizagem por reforço a uma arquitetura multiagente para suporte ao ensino de educação ambiental.** PPGCC/PUCRS, Porto Alegre, 2000. Masters Thesis.

[25] MAZZORANI, A. C.; SPAGNOLI, L. A.; MARCZAK, S. S. **Tchê: Uma Viagem pelo Rio Grande do Sul.** FACIN/PUCRS, Porto Alegre, 2001. Trabalho de Conclusão II.

[26] COMUNELLO, G.; FILHO, M. A. V.; SANTOS, T. B. **Quero-Quero Aprender Matemática.** PPGCC/PUCRS, Porto Alegre, 2001. Trabalho de Conclusão.