Player Modeling: Towards a Common Taxonomy

Marlos C. Machado, Eduardo P. C. Fantini and Luiz Chaimowicz
Computer Science Department, Federal University of Minas Gerais
Belo Horizonte, Brazil
Email: {marlos, fantini, chaimo}@dcc.ufmg.br

***Abstract*—Artificial Intelligence (AI) is gradually receiving more attention as a fundamental feature to increase the immersion in digital games. Among the several AI approaches, *Player Modeling* is becoming an important one. The main idea is to try to understand and model the player characteristics and behaviors in order to develop a better AI. This paper presents a survey of the field, discussing the main concepts and proposing a taxonomy to better organize them. We also present several game platforms that can be used by player modeling and AI researchers. We believe that compiling this information can be important to the field, specially to new researchers.**

***Index Terms*—Player Modeling, Taxonomy, Game Platforms.**

I. Introduction

For a long time, the game industry has put much of its efforts on computer graphics to increase the level of immersion of its AAA games. But in recent years, we can observe that some of the focus has started to shift to Artificial Intelligence (AI), that has been commonly relegated to a less important role. There are several reasons for this. The first one is the perception that the immersion achieved with amazing graphics can be spoiled by the behavior of *dummy* non-player characters. Another important point is that the increasing performance of the new computer architectures has allowed the use of more sophisticated AI algorithms, tightening the gap between the game industry and academic AI.

At the same time, AI researchers have been considering digital games an important platform for research. Fairclough et. al [10] argue that "computer games offer an accessible platform upon which serious cognitive research can be engaged", while Laird and van Lent suggest that computer games are the perfect platform to pursue research into human level AI [14]. Moreover, the high level of realism achieved by some games has provided us an environment similar to the real world that can be used, for example, to evaluate robotics algorithms without the costs of real robots and sensors.

At this moment is important to define the scope we refer using the word *games*. We are concerned with games as Nareyek defined: not board/card/puzzle games but the ones which demand us dealing with "optimization in real time, a highly dynamic and complex environment, incomplete knowledge" [20]. Game examples are the *Civilization*, *Starcraft*, *FIFA* and *Call of Duty* series.

Much of game AI has focused on using classical techniques such as rule based systems and graph/tree search to solve three main problems: movement, decision making and high level strategy. But as the complexity of games increase, an important research area is gaining attention: *Player Modeling*. The main idea is to try to understand and model the player characteristics and behaviors in order to develop a better AI, making games more immersive and challenging. We firmly believe that player modeling is a very promising field and many works has shared this belief [4], [15], [38]. But in spite of the growing interest in this area, there has not been an effort to join the fields' concepts and main techniques and this is essential to organize the knowledge being produced.

Our objective in this paper is to present a discussion about several player modeling concepts. We gathered information from various works in the field and generated a taxonomy that tries to organize the most important concepts. We also present some suitable game platforms that can be used to experiment player modeling techniques and AI algorithms in different game genres. This is an attempt to facilitate experimental research in this field. As far as we know there is no work that has organized a taxonomy of the field nor a list of experimental platforms and we believe this paper can be very useful to new researchers in this area.

This paper is organized as follows: next section will address the basic concepts of player modeling while the third section will discuss several approaches that are being used on this field, presenting the techniques and related works to them. Section 4 presents a discussion about several experimental platforms available for researchers. Finally, Section 5 brings the conclusion and directions for future work.

II. Player Modeling Taxonomy

In order to build a taxonomy, it is necessary to firstly adopt a common name to the field. While some papers refer to it as *Player Modeling*, others, as those from Valkenberg [37] and van den Herik, Donkers and Spronck [38] use the term *Opponent Modeling*. As Spronck and den Teuling discussed in [31], we believe the term *Opponent Modeling* is too specific since it excludes approaches that research the same tasks with different goals like, for example, cooperative work. Thus, we adopt the more general name: *Player Modeling*.

Player modeling can be defined as an abstract description of the current state of a player at a moment. This description can be done in several ways like *satisfaction*, *knowledge*, *position* and *strategy* [38].

The main goal of a game is to entertain its players, that are different from each other and may not enjoy the same challenges or possibilities of the game. When its preferences, consequently its *satisfaction*, are modeled, we may be able to

Description	Categories	Goals	Applications	Methods	Implementation
Knowledge	Online Tracking	Collaboration	Speculation in Search	Action Modeling	Explicit
Position	Online Strategy Recognition	Adversarial	Tutoring	Preference Modeling	Implicit
Strategy	Off-line Review	Storytelling	Training	Position Modeling	
Satisfaction			Substitution	Knowledge Modeling	

TABLE I
PLAYER MODELING TAXONOMY SUMMARY

adapt the gameplay to each player. This is called *satisfaction modeling*.

More related to agents' artificial intelligence, we may want to model the player *knowledge*, since this can be useful in several environments with imperfect information. A concrete example is found in games that have *fog of war*, in which answers to the following questions can be very useful: which part of the map the player knows? Does it know our position? In a game with constant evolution, which evolution level it has already achieved? All these questions may be answered with *knowledge modeling*.

Similarly, we can model the player movement. Once we are in a partially observable environment, the *position* of other players is generally an important information since it can guide your strategy or actions. This is generally called *position modeling*.

A higher-level modeling is the *strategy modeling*, which intends to interpret the player actions and relate them with game goals, *i.e.*, we abstract low-level actions seeking a high-level goal. For example, if we want to know our adversary aggressiveness, we will not be concerned with particular actions but with its strategy along the game.

These different descriptions lead us to different levels or categories of player modeling use. The lowest level is the *Online Tracking* which is concerned in predicting immediate future actions. The next level (*Online Strategy Recognition*) is related to strategy recognition as it involves the identification of a set of actions as a higher level objective or strategy. Finally, the *Off-line Review* is the evaluation of a game log, after its finish. This last level is what many professional players do when they are "studying" their adversaries for a game. Laviers et. al. [15] discuss these three topics and argues that *Online Tracking* is used for single players while *Online Strategy Recognition* can be applied to entire teams. This is true since there is not a granularity for team immediate actions but it is important to note that we can recognize strategies for individual players.

As mentioned in the previous section, we firmly believe that *Player Modeling* is a suitable approach to improve the artificial intelligence of non-player characters (NPCs) in games. This improvement can be seen in different types of goals for the NPCs, that can be divided in three main sets: to collaborate with the human players, be their opponents, or to be neutral to them but part of the story.

The first set, related to the collaborative agents, is very difficult because human players have expectations when being aided by non-player characters. These expectations are related to their actions: frequently, humans players are not able to act properly because the non-player characters will not behave accordingly as an unique team. Most of the games implement agent coordination and collaboration through basic orders such as "Attack", "Patrol" and "Hide". The main challenge would be to make these agents act autonomously according to the player behavior, without the need of specific orders.

The second set, the modeling of opponents, has motivation even on literature from centuries ago: Sun Tzu once said "Know your enemy and know yourself and you can fight a thousand battles without disaster" [36]. In addition to the advices of a historical general, it is very common for human players to study their adversaries before a match. Kasparov, the great chess player, is an example [4]. Modeling opponents is a fundamental aspect in making games more immersive and challenging, so this is where most of the works in player modeling focuses.

One last possible goal developers can be concerned with is storytelling. In complex games, not all agents are allies or enemies. They can be neutral to players, being part of the scenario and interacting with them to help advancing the plot (in a medieval world not all people are warriors or mages, there are common people that should make the story more immersive). Many times this interaction is offered to improve the storytelling of a game since once we model the player we may adjust all neutral agents to act properly to him. Artificial emotions, for example, can be used here.

In a lower abstraction level we can list, as discussed in [38], four main applications to player modeling: *speculation in heuristic search*, *tutoring and training*, *non-player characters* and *multi-person games*. We renamed and redistributed them in terms of importance and generality. We renamed *speculation in heuristic search* to *speculation in search* and we split *tutoring and training* as two different applications. Finally, we grouped *non-player characters* and *multi-person games* as *substitution* application. Several other applications can be listed but we believe that this four cover a satisfactory spectrum of them.

Speculation in search is generally applied to games in which Artificial Intelligence is more related to search in game trees, generally for adversarial goals. Depending on the game complexity it may be infeasible to check every possibility and even pruning techniques such $\alpha - \beta$ are not enough. In these scenarios, we may use player modeling to create a bias which helps the search heuristics. Carmel and Markovitch were one of the first to investigate this possibility [4].

The collaborative goal can be expressed as the use of player modeling to assist players. This can be done with tutoring, when a non-human agent teaches the player (the player modeling is important because this tutoring process can focus on the player preferences) or training, with the presentation of challenges suited to the player characteristics (weakness, style or strategy, for example). Its behavior is important because if the non-player characters do not act properly the game will no longer be interesting to the player [29].

The final main application to player modeling is in multi-person games. The objective is to allow non-player characters substitute human-players in multi-player games, even mimicking their behavior. It does not matter if the non-player characters will be allies or enemies, they must be able to replace the player to keep the previous game balance. Most of the games does not have this approach and its gameplay may be impaired by players that are not able to play the whole game.

We can also divide the player modeling field in more specific methods, which are closely related to its description purpose. Spronck and den Teuling [31] mentioned that most of the research in player modeling is done with *action models*, that are an attempt to model players' activities in a way that makes possible to predict the next player's action (*Online Tracking*). Indeed many works follow this line and [26] is a classical example since it tries to predict if an agent will declare war against other in the *Civilization IV* game. Another important work is [13] that anticipated actions in the *Quake II* game.

Spronck and den Teuling define *preference modeling* in [31] as the modeling of the "player desires to accomplish or experience in the game, and to what extent he is able to do that". This is a very precise definition and, in fact, is concerned to the player's satisfaction. Some of the recent works that addressed this problem are [17] and [22].

The last two listed methods are *positioning* and *knowledge modeling*. The first one attempts to obtain relevant information about players location while the second tries to model the player knowledge itself, it means, what he already knows.

Positioning modeling can be better defined as the attempt to predict non-player characters positions on games with imperfect information (*fog of war*, for instance). This is a valuable information because in general, the knowledge of a player position gives a tactical advantage in a game, as previously discussed here. This approach may be seen as an attempt to present smart non-player characters which does not break game rules (ignoring *fog of war* for example), a common used resource as Laird and van Lent already discussed [14]. Valkenberg worked with this problem trying to foresee players position in *World of Warcraft* [37], despite the fact it did not have much success, the problem he worked was a classical one. A more successful approach was presented in [12] in the *Counter Strike: Source* game.

Knowledge modeling, on the other hand, tries to "predict" the other players knowledge, humans or not. Cunha and Chaimowicz in [5], for example, developed an advice system to RTS players that implemented a predictor of the technological level of an agent based on its units. It was done in the game *Wargus* modeling a reverse path in the dependency tree of the game, deriving what are the technologies known by the enemy from the observation of existent buildings and units.

Once we defined some of the *player modeling'* subsets related to goals, applications, research areas, among others, we may finish this section with the lower abstraction level of discussion: the implementation. Two approaches can be highlighted: *explicit* and *implicit*.

Spronck says that "An opponent *[player]* model is explicit in game AI when a specification of the opponent's *[player's]* attributes exists separately from the decision-making process" [30]. Thus, an explicit player model is separated from the main source code and generally implemented through scripts or XML files. On the other hand, in *implicit* approaches, the attributes are generally embedded and diluted in different parts of the code, which makes the task of identifying and describing these attributes more difficult.

The main components discussed in this section are summarized in Table I. We believe that they can be a start point towards a more comprehensive taxonomy of this area. In the next section we review and discuss some common AI techniques used in Player Modeling.

III. Common Techniques

Several AI techniques have been studied and used for Player Modeling both in the academy and in the game industry. This section presents the most important techniques and some works in which they have been applied.

One of the first works that presented a kind of taxonomy of player modeling techniques was [38] and one of its main concerns was the player model representation. This is not a simple question since van den Herik, Donkers and Spronck [38] affirm that "the internal representation of an opponent model *[player model]* depends on the type of knowledge that it should contain and the task that the opponent *[player]* should perform". The authors also discuss several techniques that make this possible: *evaluation functions*, *neural networks*, *rule-based models*, *finite-state machines*, *probabilistic models*, and *case-based models*. We will resume this discussion and will also add other possibilities such as *genetic algorithms/programming* and *Monte-Carlo search*.

Evaluation functions are concerned in understanding and modeling how players evaluate the game states. Once we have modeled the way a player sees the game, we can better predict its future actions, using search algorithms for example. A specific evaluation function can be seen as a player model. This approach is proposed in [4] where the authors define player model as a "recursive structure consisting of the opponent's evaluation function (...)". Their method uses this model to set the utility function of a node in a tree search. An example of this concept in storytelling (drama management) is presented in [25] that modeled the player satisfaction with a function that is obtained by its behavior.

Other technique, generally more complex than evaluation functions, are *neural networks*. They are widely used in different applications and can also be used in games. A neural network can be seen as a "black box" that maps inputs to outputs. This black box is implemented through a set of interconnect nodes (neurons) that are activated according to their weighted inputs. A training process is used to adjust the weights to the desired objective. Some of the works which used this technique are [22] that collected several gameplay statistics and used Single-Layer and Multi-Layer Perceptrons (special types of neural networks) to develop a model of player satisfaction, and [17] that used Self-Organizing Maps to automatically generate player models groups and then use Single-Layer Perceptrons to fit a non-linear function that relates the game statistics and these groups. Neural networks are more established than other techniques discussed here and have also been used in commercial games such as *Creatures* and *Black & White* [3].

Rule-based models consist of a set of conditions that, when satisfied, generate a series of actions. It's very easy to be implemented and tested and maybe for this reason it is widely used in the game industry. *Finite-State Machines* (FSMs) can be seen as a variation of Rule-based models. Fairclough et. al. [10] affirmed that FSMs were the most used technique in FPS games and a more recent work [21] corroborated it citing several authors who affirm that many games implement non-player characters artificial intelligence with Hierarchical Planners, Hierarchical FSMs and Behavior Trees.

A variation of FSMs are *Probabilistic Models* since they are FSMs augmented with state transition probabilities on the edges. They are useful in games with imperfect information, such as Poker for example [38]. We can also list *Fuzzy State Machines* (FuSM) that can be seen as a generalization of FSMs, in which transitions are modeled using fuzzy rules instead of boolean logic. They receive a degree of membership since a FuSM can be at more than one state in a given moment.

Case-Based Models are a different approach from those being discussed. It consists in a case base with situations and actions. It is used with a query to the case base and the selection of an appropriate action, based on the actions of the selected cases. It is useful because it is easily updated and expanded with new cases, as discussed by [38].

Different techniques from those listed in [38] are *genetic programming (GP)* and *genetic algorithms (GA)*, which are inspired by evolution theory: solutions (or evaluation functions, among others) are modeled as chromosomes and iteratively evolve these chromosomes applying operators like mutation and cross over to explore the search space trying to improve the solution. Its intuition is to combine the best solutions (chromosomes) in order to generate better ones, also applying a random factor (mutation) to introduce new solutions. There are several works that use GP and GA, not necessarily to model opponents, but somehow to improve their behavior. Spronck and Posen [32], for example, obtained a high level strategy for the RTS game *Wargus* with *genetic algorithms* where the strategy was defined by the sequence of buildings that are constructed during the game.

Another interesting approach is *Monte-Carlo Tree Search (MCTS)* that is based on Monte-Carlo simulations: it performs game simulations that are used to estimate the values of game states and actions. This information is used to progressively improve the quality of the simulations. Ponsen et al. [24] have used MCTS in poker, to allow the search algorithm to focus on relevant parts of the game tree. They also mentioned that MCTS is recognized "as the current paradigm for computer Go". Branavan et al. [1] developed a different work using MCTS in the TBS game Civilization II (they also used neural networks). Their approach consisted in using *value function approximation* locally in time to the current set of possibilities. They also automatically extracted domain knowledge from texts and this domain was responsible for the agents behavior.

A whole new area that is receiving increasingly attention is machine learning. Some of the techniques we already discussed can be seen, in different levels, as learning algorithms. The discussion about learning algorithms generates an obvious question: *what can we learn?* van den Herik, Donkers and Spronck [38] tries to answer this question with three concepts: evaluation functions, probabilistic models and opponent behavior. It is interesting to observe that some of these concepts have already been discussed above. This is not a closed and well defined list and other possibilities can also be listed as learning player preferences or player decision trees. Despite not being discussed in details in this paper, learning algorithms applied to player modeling is a very interesting area and can be the subject of another entire paper.

It is important to remark that there are many other approaches and it is impossible to list and discuss all of them. Game development demands knowledge from several fields of computer science and contributions may come from everywhere. An example of this is [21], where the authors made an analogy between compilers and behavior. They modeled the cognitive system of a character as a compiler, using concepts like *token*, *lexical analysis* and *syntactic analysis*. We also have more traditional approaches that have not been discussed here like Input-Output Hidden Markov Model used by [11] to recognize players goals or Multi-agent approaches intending to improve other techniques results like MCTS [16].

In the next section we present some platforms that allow us to experiment and validate research works in artificial intelligence for games. Although these platforms are not exclusive for player modeling, we discuss some player modeling research topics for some game genres.

IV. Some Platforms

As previously discussed, sometimes it is very difficult to experiment and validate new AI approaches in games. Part of this difficulty is due to the fact that we need an effective and complete game to validate our hypothesis. To validate any artificial intelligence algorithm in games we have two main paths: (i) build our own game; (ii) integrate our algorithms with some existing game. Both are not trivial to do.

The first alternative, build our own game, may not be interesting because it is extremely time-consuming and the result will be a prototype, probably with simpler rules and environments that can be useful but will not have the same appeal as a complete game. Furthermore, for scientific reasons, despite a possible availability of the source code, the comparison of different algorithms, of different researchers, become very hard. Finally, since it is a prototype, there will not be expert players for tests with this platform. On the other hand, building our own game facilitates the implementation of the desired algorithms and the adaptation of the code to test different techniques. This approach is frequently used and some examples are presented in [11], [17] and [21].

The use of a pre-built game, maybe a commercial one, is more attractive for us. We can subdivide this approach in two different game types: (i) open-source games and (ii) games which supports scripted AI (explicit or implicit implementations). The following sections will try to list some suitable game platforms and interfaces for game AI research. This discussion is also available on our research group website[1] where we are able to present more details about each platform.

A. Action Games

This game genre offers many possibilities to researchers since strategy, knowledge, position and satisfaction can be modeled in this type of game. It allows several applications as tutoring and substitution of human players.

Some of the available platforms that we judge interesting to list are the following:

1) Counter Strike: Source (CS:S): it is a FPS multiplayer game that was published by Valve Software in 2004. Its gameplay consists in two opposing teams (Terrorists and Counter-Terrorists) that are in constant military combat. A common goal for every match is to kill your enemies, besides team-specific tasks as enable/disable bombs and arrest/rescue hostages. Valve Software has made the Source Engine SDK available: it allows common players create particular game mods, which can be customized in configuration files and source code (in C++). The computer controlled agents (bots) in CS:S are server-side. An example of research done in this platform can be found in [12].

2) Doom, Quake, Wolfstein 3D: these are FPS games whose source code has been made available by its creators. They are grouped because have very similar characteristics and they are produced by the same company, *id Software*. They provide these games in two different ways: the complete source code or through SDKs that contain the main source-code of the game. In this case, it is possible to make changes and compile the code generating DLLs that shall replace the older ones. *Quake 4* is the most recent game that has been made available in SDK form, being released in 2006. Some works that used *Quake II and III* as testbed platforms are [13], [33], [34], [35] and [28]. The source-codes of these titles are written in C/C++ and are available on the FTP link[2] of the company.

B. Adventure Games

Adventure Games are more limited to player modeling research since they do not have physical challenges or teams, in general. In spite of that, its interactive story and puzzle-solving allows research in storytelling and collaboration. We could also model players satisfaction adjusting puzzle difficulties according to its satisfaction. An interesting validation platform is the Adventure Game Studio, presented next.

1) Adventure Game Studio (AGS): this is a drag and drop tool that allows the creation of adventure games similar to the classic ones from Sierra and LucasArts studios of the 90's. It is free to non-commercial use. Several game aspects can be customized as dialogs, graphics, scripts in Java/C#, etc. The AGS manages some game parts as menus, load/save functions, pathfinding and scrolling rooms. Moreno-Ger et. al. refers to this platform as an option to creation of adventure games [18].

C. Platform Games

This game genre allows research in different aspects of satisfaction modeling, for example, collaboration and adversarial goals. Some of the most well-succeeded games in history are in this genre and we decided to list maybe the two most significant: Mario and Sonic.

1) Infinite Mario Bros: Super Mario Bros is an 2D action/platform game released in 1985 by Nintendo. The game goal is to control the game protagonist, Mario over the Mushroom Kingdom, fighting against Bowser, the villain, to save princess Peach. The Infinite Mario Bros is an open source project that mimics Super Mario Bros. It was developed in Java and is used as testbed in papers like [22] and [27]. Also, an Infinite Mario Bros adaptation is used in the Mario Artificial Intelligence Championship[3]. A similar open source project is the Secret Maryo Chronicles, written in C++. In both projects the game artificial intelligence shall be modified directly on the source code.

2) Open Sonic: Sonic the Hedgehog is a 2D action/platform game released by SEGA in 1991. Its objective is to gather items and to defeat non-player characters while the player advances through the map using its skills and reflexes. Open Sonic is an open source project based on the game Sonic the Hedgehog. It allows the player to control three different characters with specific functionalities to interact with the scenario and defeat its enemies. New items, enemies and bosses may be created with Object Scripts or changing its source code, written in C.

D. Role Playing Games

RPG games provide several research possibilities. Most of the topics discussed in this paper can be, somehow, studied in this genre since it provides a rich environment that allows interactive storytelling, strategy definitions and can be played both in single and multiplayer modes, in a cooperative or adversarial fashion. This environment is very interesting for AI researchers and this is confirmed by the number of available platforms and published papers.

[1]http://www.j.dcc.ufmg.br/platforms.html

[2]ftp://ftp.idsoftware.com/idstuff/

[3]http://www.marioai.org/

1) Baldur's Gate I and II: this is a RPG game developed by BioWare, with its first version being released in 1998. It was developed following the Forgotten Realms RPG rules. In this game the player evolves over the plot chapters through dialogs, quests and battles. The versions I and II were developed with the game engine Infinity which has some unofficial editors for the creation of mods, such as Near Infinity, Dialog Checker and Weidu. These editors allow dialog, sounds, GUI, and items customizations, besides AI scripts. An interesting work that used this platform is [23].

2) Neverwinter Nights: this is a 3D RPG game that was also published by BioWare, in 2002. It is based on the third edition of the RPG D&D (Dungeons and Dragons). As in Baldur's Gate, the gameplay consists in evolving the player over time by completing quests, gathering items and interacting with non-player characters. Its source code is private but BioWare published an official tool called Aurora Neverwinter Toolset that allows the modification of several games aspects using the script language NWScript. With this tool, it is possible to create game modules with environments and scene objects with customized parameters, as non-player characters, items, waypoints and triggers. There is a great number of works that used this platform and some examples are [6], [23], [39].

3) RunUO: Ultima Online Server Emulator: Ultima Online (UO) was one of the first Massively Multiplayer Online RPG Games (MMORPGs). It was released in 1997 by Origin Systems. Its fantasy universe is based on the Ultima series and it has had several expansions since its release. Its official servers are still online nowadays, despite its hosting being specific to some countries. UO fans developed several server emulators that allow the customization of several game aspects. RunUO is one of these emulators: it receives connections from all UO official clients besides an unofficial, open source client called Iris 2. It was developed in C# and allows a complete game customization.

E. Simulation Games

Simulation Games are useful for player modeling research, not only for entertainment purposes but also for validation and experimentation of models that run outside games. It offers a wide range of possibilities since it can, in theory, simulate any world event. Its possibilities shall be analyzed case by case.

1) Flight Gear: is an open-source 3D flight simulator. The project goal is to offer a sophisticated framework for realistic flight simulations in private projects, both commercial and academic, as in [9]. Several airplanes, with realistic controls, and different scenarios based on the most recent data from The Shuttle Radar Topography Mission (SRTM) are available. We can modify its source code, written in C++, or its 3rd party extensions.

2) The Open Racing Car Simulator (TORCS): is an open source 3D race simulator that allows disputes between real pilots or computer-controlled pilots. The opponents artificial intelligence can be written in C or C++ directly in the platform source code. It has a good 3D graphical quality and an accurate physics, being used as race game or as artificial intelligence platform, as done in [19].

F. Sports Games

Sports games are an excellent environment for experimentation of player modeling. They offer several important problems as teamwork behaviors, strategy adaptation to the players style or the tutoring and training possibilities. Unfortunately we could not find many research platforms for this promising environment.

1) Super Tux Kart: this is an open source kart racing game. Its game mechanics consist in the racing against real pilots or computer-controlled agents, gathering special items along runway. It is similar to Super Mario Kart, with 3D graphics and physics engine. New 3D player models can be inserted from XML add-ons and the AI customization for them can be written in C++ in the source code. It has been already used in research works such as [27].

G. Strategy Games (RTS and TBS)

This game genre is certainly one of the most challenging for artificial intelligence. We have several research topics, many of them listed in [2]. As RPG games, the challenging environment with a large number of agents and different game strategies attracts several researchers. This is evidenced by the great number of platforms and published papers.

1) Civilization IV: this is a turn-based game released in 2005. There are two different interfaces to edit this game: XML and the source-code. The XML interface offers the possibility of configuring several parameters of the game, as the agents "flavors" [31] while the source-code interface is similar to the interface presented in Quake 4: it is a SDK that allows people to change the source code of the game and compile it generating a DLL that replaces the original one. This is one of the most complex strategy games, with several possibilities for development, game ending, agents behavior, among others. To illustrate this statement, *Civilization IV* presents six different victory conditions [7]. Due to its complexity and multiple possibilities, it has been largely used in game AI research: [7], [26], [31], [41] are some of the authors that have used this platform to validate their theories. Besides Civilization IV we also have other versions and variants of the games of the Civilization series [41]. One of the most popular variants is the open source version of Civilization II called FreeCiv. Branavan, Silver and Barzilay have recently used this platform in their research [1]. C-Evo is another open-source variant of Civilization II and, as FreeCiv, is deeply discussed in [41].

2) ORTS: this is a RTS engine implemented in a client/server architecture with 2D or 3D graphics. As any RTS platform, it is largely used in research works dealing with real-time artificial intelligence such as pathfinding, imperfect information treatement, scheduling and planning. Users are responsible for defining the game characteristics and this is done by a script that describes the combat units, structures and available interactions. The server side loads these configurations and waits the clients connection. The units artificial intelligence is written in C++ in the client side. Once the user is connected to the server it can send commands to its objects. This platform was used from 2006 to 2009 in the Open RTS Game AI Competition promoted by the Artificial Intelligence for Interactive Digital Entertainment Conference (AIIDE). It is described in details in [2].

3) Starcraft I and II: StarCraft is a commercial real time strategy game with fictional military characteristics. StarCraft II Wings of Liberty was developed and released in 2010 by Blizzard Entertainment. Its source code is not available but the StarCraft II Editor is an official tool to game modding. This tool does not require programming knowledge and allows the customization of maps, triggers, text, buildings, weapons and combat units. Other option to game editing is the Galaxy Scripts (based on C) that can be imported by this editor. The scripts are edited with MPQEditor (unofficial). StarCraft Brood War is an older game version, released in 1998. There is an unofficial API for it called BWAPI that allows the creation of AI modules in C++ that are inserted in the game by loaders such as Chaoslauncher and MPQDraft. BWAPI is used since 2010 in the StarCraft AI Competition organized by the AIIDE. One of the recent works that have used this platform is [40].

4) Wargus: This RTS game is a mod of the game Warcraft II, released by Blizzard Entertainment in 1995. It is an open source project organized in two levels: the core, with its basic game functions (engine Stratagus) done in C++ and a higher level, the game logic, implemented in Lua. Two of the many papers that used Wargus are: [5], [32].

H. *Off-line Review*

Player Modeling research does not always require game modifications. A whole category is independent of the game code: Off-line Review. This category requires only the game monitoring and this can be done even if we are not able to edit the source code (or a script). A work that exemplifies this is [8] that used the EIDOS Suite Metrics to monitor the game Tomb Raider: Underworld.

The EIDOS Suite Metrics[4] is an instrumentation system which records game metrics from EIDOS Studios games. Game metrics are transmitted to an SQL-server by an Extract, Transform and Load (ETL) process through which they are logged as sequences of events. These captured data can now be processed and extracted, creating reports for the interested parties within the game development process (e.g. game design, production, quality assurance and marketing).

V. Conclusion

In this work, we proposed and discussed a taxonomy for player modeling research gathering and organizing information from several different sources. This taxonomy was summarized in Table I, which tries to characterize the most important topics in this area. We also expanded a discussion from [38] about the most common techniques used in this field presenting other techniques and several relevant recent works that have used some of them. Finally, we did an analysis of player modeling research possibilities in several game genres and also listed suitable game platforms for experimentation, discussing their main characteristics. We consider that this is an important information since it is not widely available for new researchers, which can have much more difficult in the area for not knowing these possibilities. Based on the concepts and techniques organized here and also on the listed platforms, we believe that this work will be a very useful text to new player modeling researchers.

There are several works that can be developed from this one, which presented the general concepts of player modeling. A more quantitative work can be developed, using the taxonomy proposed here, to analyze the current research in player modeling, classifying the related papers in terms of platforms, techniques and description purposes. Also, as we previously said, a complete new analysis can be done in terms of learning algorithms. First of all, we may correlate the main learning techniques to the main problems of the field. We could also build a survey of the relevant works that have used these techniques, analyzing its use in the game industry. Finally, another possible work related to this paper is an analysis of the current use of all the techniques discussed here (or even a higher set, with learning, for example) in commercial games. A work that tried to do this was [38] but its analysis was restricted to few games. This would be certainly an interesting but hard work since the code of most commercial games is not available for academic research.

[4]This is a proprietary software but we judged valid to describe its approach

Acknowledgments

This work is partially supported by CAPES, CNPq and Fapemig.

References

[1] S. Branavan, D. Silver, and R. Barzilay. Non-Linear Monte-Carlo Search in Civilization II. In *International Joint Conferences on Artificial Intelligence (IJCAI) (To Appear)*, 2011.

[2] M. Buro and T. M. Furtak. RTS games and Real-Time AI Research. In *In Proceedings of the Behavior Representation in Modeling and Simulation Conference (BRIMS)*, pages 51–58, 2004.

[3] J. Byrne, M. O'Neill, and A. Brabazon. Optimising Offensive Moves in Toribash Using a Genetic Algorithm. In *Proceedings of the Sixteenth International Conference on Soft Computing (MENDEL)*, 2010.

[4] D. Carmel and S. Markovitch. Learning Models of Opponent's Strategy in Game Playing. In *Proceedings of The AAAI Fall Symposium on Games: Planing and Learning*, pages 140–147, 1993.

[5] R. L. F. Cunha and L. Chaimowicz. An Artificial Intelligence System to Help the Player of Real-Time Strategy Games. In *Proceedings of SBGames - Computing Track - Full Papers*, pages 74–81, 2010.

[6] M. Cutumisu and D. Szafron. A Demonstration of Agent Learning with Action-Dependent Learning Rates in Computer Role-Playing Games. In *Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE)*, 2008.

[7] F. den Teuling. Player Modelling in Civilization IV. MSc Thesis, Tilburg University, 2010.

[8] A. Drachen, A. Canossa, and G. N. Yannakakis. Player Modeling using Self-Organization in Tomb Raider: Underworld. In *Proceedings of the Fifth International Conference on Computational Intelligence and Games (CIG)*, pages 1–8, 2009. IEEE Press.

[9] P. Ehlert, Q. Mouthaan, and L. Rothkrantz. Recognising Situations In A Flight Simulator Environment. In *Proc. of 3rd Int. Conf. on Intelligent Games and Simulation (GAME-ON)*, pages 165–169, 2002.

[10] C. Fairclough, M. Fagan, B. M. Namee, and P. Cunningham. Research Directions for AI in Computer Games. In *Proceedings of the Twelfth Irish Conference on Artificial Intelligence and Cognitive Science*, pages 333–344, 2001.

[11] K. Gold. Training Goal Recognition Online from Low-Level Inputs in an Action-Adventure Game. In *Proceedings of the Sixth Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE)*. AAAI Press, 2010.

[12] S. Hladky and V. Bulitko. An Evaluation of Models for Predicting Opponent Positions in First-Person Shooter Video Games. In *Proceedings of IEEE 2008 Symposium on Computational Intelligence and Games (CIG)*, pages 39–46, 2008.

[13] J. E. Laird. It Knows What You're Going to Do: Adding Anticipation to a Quakebot. In *Proceedings of the Fifth International Conference on Autonomous Agents (AGENTS)*, pages 385–392, 2001. ACM.

[14] J. E. Laird and M. van Lent. Human-Level AI's Killer Application: Interactive Computer Games. In *Proc. of the 17th National Conference on Artificial Intelligence and 12th Conference on Innovative Applications of Artificial Intelligence*, pages 1171–1178. AIII Press, 2000.

[15] K. Laviers, G. Sukthankar, D. W. Aha, and M. Molineaux. Improving Offensive Performance Through Opponent Modeling. In *Proceedings of the Fifth Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE)*, 2009.

[16] L. S. Marcolino and H. Matsubara. Multi-Agent Monte Carlo Go. In *Proceedings of Tenth International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 21–28, 2011.

[17] H. P. Martínez, K. Hullett, and G. N. Yannakakis. Extending Neuroevolutionary Preference Learning through Player Modeling. In *Proceedings of IEEE 2010 Symposium on Computational Intelligence and Games (CIG)*, 2010.

[18] P. Moreno-Ger, J. L. Sierra, I. Martínez-Ortiz, and B. Fernández-Manjón. A Documental Approach to Adventure Game Development. *Science of Computer Programming*, 67:3–31, June 2007.

[19] J. Muñoz, G. Gutierrez, and A. Sanchis. Controller for TORCS Created by Imitation. In *Proc. of the 5th International Conference on Computational Intelligence and Games (CIG)*, pages 271–278, 2009. IEEE Press.

[20] A. Nareyek. Computer Games - Boon or Bane for AI Research? *KI*, 18(1):43–46, 2004.

[21] J. Orkin, T. Smith, and D. Roy. Behavior Compilation for AI in Games. In *Proceedings of the Sixth Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE)*, pages 162–167. AAAI Press, 2010.

[22] C. Pedersen, J. Togelius, and G. N. Yannakakis. Modeling Player Experience for Content Creation. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(1):54–67, 2010.

[23] I. S.-K. Pieter Spronck and E. Postma. Online Adaptation of Game Opponent AI. In *Proceedings of the Fourth International Conference on Intelligent Games and Simulation (GAME-ON)*, pages 93–100, 2003.

[24] M. Ponsen, G. Gerritsen, and G. Chaslot. Integrating Opponent Models with Monte-Carlo Tree Search in Poker. In Proceedings of Interactive Decision Theory and Game Theory Workshop at the Twenty-Fourth Conference on Artificial Intelligence (AAAI), 2010.

[25] D. L. Roberts, C. R. Strong, and C. L. Isbell. Using Feature Value Distributions to Estimate Player Satisfaction Through an Author's Eyes. In *In Proceedings of the AAAI Fall Symposium on Intelligent Narrative Technologies*, 2007.

[26] M. Rohs. Preference-based Player Modelling for Civilization IV. B.Sc. Thesis. Maastricht, The Netherlands: Faculty of Humanities and Sciences, Maastricht University, 2007.

[27] S. Ross and D. Bagnell. Efficient Reductions for Imitation Learning. *Journal of Machine Learning Research - Proc. Track*, 9:661–668, 2010.

[28] P. S. Sander Bakkes and E. Postma. Best-Response Learning of Team Behaviour in Quake III. In *International Joint Conferences on Artificial Intelligence Workshop on Reasoning, Representation, and Learning in Computer Games*, pages 13–18, 2005.

[29] B. Scott. *The Illusion of Intelligence*, pages 16–20. Charles River Media, Inc., Hingham, MA, USA, 2002.

[30] P. Spronck. A Model for Reliable Adaptive Game Intelligence. *Int. Joint Conferences on Artificial Intelligence Workshop on Reasoning, Representation, and Learning in Computer Games*, pages 95–100, 2005.

[31] P. Spronck and F. den Teuling. Player Modelling in Civilization IV. In *Proceedings of the Sixth Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE)*, pages 180–185. AAAI Press, 2010.

[32] P. Spronck and M. Ponsen. Automatic generation of strategies. In S. Rabin, editor, *AI Game Programming Wisdom 4*, pages 659–670. Charles River Media, Hingham, MA, 2008.

[33] C. Thurau, C. Bauckhage, and G. Sagerer. Combining Self Organizing Maps and Multilayer Perceptrons to Learn Bot-Behaviour for a Commercial Game. In *Proceedings of the Fourth International Conference on Intelligent Games and Simulation (GAME-ON)*, pages 119–, 2003.

[34] C. Thurau, C. Bauckhage, and G. Sagerer. Learning Human-Like Movement Behavior for Computer Games. In *From Animals to Animats 8: Proceedings of the Eighth International Conference on Simulation of Adaptive Behavior (SAB)*, pages 315–323, July 2004.

[35] C. Thurau, T. Paczian, and C. Bauckhage. Is Bayesian Imitation Learning the Route to Believable Gamebots? *Int. Journal of Intelligent Systems Technologies and Applications*, 2(2/3):284–295, 2007.

[36] S. Tzu. *The Art of War*. Tribeca Books, 2011.

[37] A. J. J. Valkenberg. Opponent Modelling in World of Warcraft. B.Sc. Thesis. Maastricht, The Netherlands: Faculty of Humanities and Sciences, Maastricht University, 2007.

[38] H. van den Herik, H. H. L. M. Donkers, and P. H. M. Spronck. Opponent Modelling and Commercial Games. In *Proc. of IEEE 2005 Symposium on Computational Intelligence and Games (CIG)*, pages 15–25, 2005.

[39] G. van Lankveld, S. Schreurs, and P. Spronck. Psychologically Verified Player Modelling. In *Proceedings of the Tenth International Conference on Intelligent Games and Simulation (GAME-ON)*, pages 12–19, 2009.

[40] B. G. Weber and M. Mateas. A Data Mining Approach to Strategy Prediction. In *Proc. of the 5th International Conference on Computational Intelligence and Games (CIG)*, pages 140–147, 2009. IEEE Press.

[41] S. Wender and I. Watson. Using Reinforcement Learning for City Site Selection in the Turn-Based Strategy Game Civilization IV. In *Proceedings of IEEE 2008 Symposium on Computational Intelligence and Games (CIG)*, pages 372–377, 2008.

Marlos C. Machado is a graduate student in the Computer Science Department at Federal University of Minas Gerais. His research interests include machine learning and artificial intelligence for games, in particular player modeling. The work described in this paper is related with his Master's research.

Eduardo P. C. Fantini is a graduate student in the Computer Science Department at Federal University of Minas Gerais. His research interests include multi-agent coordination, chatterbots and general artificial intelligence for games.

Luiz Chaimowicz received his M.Sc and Ph.D. in Computer Science from the Federal University of Minas Gerais, Brazil in 1996 and 2002 respectively. From 2003 to 2004, he held a Postdoctoral Research appointment with the GRASP Laboratory at University of Pennsylvania. He is currently an Associate Professor in the Computer Science Department at the Federal University of Minas Gerais, Brazil. Dr. Chaimowiczs research interests includes artificial intelligence for games and mobile robotics, more specifically on multi-agent coordination and path planning.