Modelling Human Expert Behaviour in an Unreal Tournament™ 2004 Bot

A.M. Mora, F. Aisa, R. Caballero, P. García-Sánchez,
P.A. Castillo, J.J. Merelo, and P. De las Cuevas

Departamento de Arquitectura y Tecnología de Computadores.
Universidad de Granada (Spain)
{amorag,pgarcia,pedro,jmerelo,paloma}@geneura.ugr.es,
francisco_aisa@hotmail.com, rcabamo@gmail.com

Abstract. This paper presents a deep description of the design of an autonomous agent (bot) for playing 1 vs. 1 dead match mode in the first person shooter Unreal Tournament™ 2004 (UT2K4). The bot models most of the behaviour (actions and tricks) of an expert human player in this mode, who has participated in international UT2K4 championships. The Artificial Intelligence engine is based on two levels of states, and it relies on an auxiliary database for learning about the fighting arena. Thus, it will store weapons and items locations once the player has discovered them, as a human player could do. This so-called expert bot yields excellent results, beating the game default bots in the hardest difficulty, and even being a very hard opponent for the human players (including the expert).

1 Introduction

Autonomous agents in videogames (so-called *bots*) have tried to behave as human players from their emergence in the first years of the nineties. They normally model a part of a human expert knowledge regarding the game, in order to become a competitive opponent to play against some other humans or bots, or lately, to cooperate with or aid the main players. They are also named non-playing characters (NPCs), classical secondary characters in conversational or rol games, but referring nowadays to the same concept as bots: an Artificial Intelligence (AI) engine which is able to perform the same actions in a game as a human player.

First Person Shooter games (FPSs) are one of the most profitable area in the study and implementation of bots. In these games the player can only see the hands and the current weapon of her character, and has to fight against enemies normally by shooting to them. They usually offer a multiplayer fighting mode placed in a limited arena. From their first years (begin of the nineties) these games have been open to the creation of bots, initially aimed to their graphical aspect, but later giving tools to implement even their whole AI (or a part).

Unreal™ , launched for PC by Epic Games in 1998, had a great success since it incorporates the best multiplayer mode to date. In addition it started an open-code philosophy to make it easier the game-modding (modification), including

with every copy of the game an editor (UnrealEd), an own programming language (UnrealScript), and a compiler, to add or change almost whatever the user desires: scenarios, items, characters, etc. The AI engine is also open, so the state-based behaviour implemented in the game characters can be changed in a number of ways, just having some critical actions restricted.

Moreover, some additional tools have arisen a few years ago, such as GameBots [1], a *mod* (new utility for the game) that allows the control of characters (bots) in the game through network connections to other programs. The game sends character's sensory information to the client program, which can decide the actions the bot will take. These actions are sent back to the game which interprets them for the bot movement, shooting, jumping, etc. It was initially released for the Unreal sequel *Unreal Tournament™ (UT)* and later implemented for *UT 2003*. Over the basis of that tool, it was later launched *Pogamut* [2], which defines an interface (using GameBots architecture) to program the bots externally using Java. It was implemented for Unreal Tournament™ 2004 (UT2K4).

These open-code possibilities and external tools are the main reasons why this game environment have been widely considered in the computational intelligence researching field [3–9].

This work is also embedded in the UT2K4 environment, and uses Pogamut for implementing a bot which we have baptised as *UT Expert Bot (E-Bot)*. Its behaviour (AI) is shaped as a Finite State Machine (FSM) [10] (based on two levels of states), that describes a complex set of rules. These rules are based on the knowledge of an UT2K4 Spanish expert player. It has been modelled according to his experience, including several tricks, as the humans players do, to achieve a better behaviour in the game.

2 Unreal Tournament™ 2004 Game Features

As previously commented, Unreal™ was a very famous FPS published in 1998 for PCs. It presented a very good single mode, but the multiplayer possibilities gave it (and still give) a great success. Thus, in 1999 it was released Unreal Tournament™, which turned the series into multiplayer-aimed games, having several arenas specifically designed for massive battles between humans and/or bots. The most successful sequel in this series was Unreal Tournament 2004 (UT2K4), due to the original weapons, the excellent design of scenarios and the famous amazing opponents' AI, based on states and events, inside a huge Finite State Machine [10] (where plenty of *states* and *substates* are present), and including several *optimised scripts* (predefined actions).

This framework inherited all the features that the first Unreal Engine offered for modders (people who add or change components in the game), such as maps, weapons, items, or even bots. Some of these features are:

– It includes a proprietary programming language, called *UnrealScript*, which combines the C and Java syntax, with some useful features, such as a garbage collector. It is object-oriented and handles implicit references (every object is represented by a pointer).

– This language includes the declaration and handling of *states*, which are the most powerful feature of the Unreal bots' AI. They model a bot status and behaviour at a time, and are defined as classes. During the game (depending on the bot location and status), the current state of the bot is changed, and the functions defined in it are performed.
– In addition to the game, a programming environment, named *UnrealEditor* is included. It makes it easier the management of the hierarchy of classes, as well as the creation of new classes which inherit from the existing ones.
– It is possible to change an existing class (making a mod) by creating another one which inherits from it, and modifying the code of the desired functions or methods inside the new class.

In addition, this new engine (Unreal Engine 2.5) fixed some of the drawbacks that UnrealScript had, such as the small number of elements in arrays, the limitations in the number of iterations in loops, or the file input/output.

Moreover there were some projects which created specific tools for interacting with UT and UT2K4 engines, such as the aforementioned Gamebots [1] and Pogamut [2]. These projects let the user to implement mods (mainly bots), using more powerful and flexible programming languages than the limited UnrealScript, like Java. The latter is the tool we have considered in this work due to its 'simplicity' and proved value in the implementation of bots (it has been widely used by several authors in the literature).

Going back to the game, the most traditional combat mode is *Death Match*, in which the player must eliminate as many enemies as possible before the match ends, avoiding being defeated by other players. Everyone has a limited number of health points, which are decreased as the character gets hurt. If the health counter goes down to 0, the player is defeated, and a *frag* is added to the last player who shot him. After being killed, the player is then respawned (usually in the same place or quite near) in the scenario. A match ends when the termination conditions (typically a limit of frags/kills or time) are reached.

In order to aid players to succeed in the match, some elements appear periodically in the arena: *weapons* (with limited ammunition, and an associated power) to defeat the enemies , and *items*, that provides the player with some advantages, such as extra health, high jump, invisibility or ammunition, for instance.

Dead Match mode is usually played by a number of players/bots up to 32 in the recent games of the series, but there is a very interesting variation; the *1 vs 1 Death Match* battle. It is considered in the official competitions of UT2K4 for human players and presents some specific rules and constraints: there are some items forbidden (U-Damage), weapons are not respawned, and the match runs for 15 minutes, not for a number of frags (number of enemies killed).

3 State of the Art

In the nineties, bots started to be widely used in FPSs, when Quake™ became the most successful game in including user-created autonomous characters. It presented not only the option of playing against machine-controlled bots, but

also the possibility to modify them (just in appearance or in a few other aspects), or create new ones by means of a programming language named QuakeC, which unfortunately was strictly limited and hard-constrained.

Unreal™ series appeared some years later, being (as aforementioned) the first game in including an easy programming environment and a more powerful language, thus plenty of bots were developed (just a few applying metaheuristics or complex AI techniques), and most of these bots were based on predefined hard-coded scripts.

One of the most fertile areas inside computer games and AI, is devoted to get improvements on some components of the characters' AI. These studies appeared more than a decade ago We started our research in this field in 2001, publishing our results in national conferences [3] (in Spanish). We applied a Genetic Algorithm to improve the parameters in the bots AI core (as later other authors did [11]), and to change the way of controlling the bots by automatically redefining, by means of Genetic Programming (GP), the standard set of rules of the main states. Several other evolutionary approaches have been published, such as [12], where evolution and co-evolution techniques have been applied, [7] which applies an evolutionary rule-based system, or the multi-objective approach presented in [9] which evolves different specialised bots. The two latter have been developed inside the Unreal Tournament™ 2004 environment (UT2K4).

Also in the first years, studies involving other techniques arose, such as [13], where the authors used self-organizing maps and multilayer perceptrons, or as [14], which applied machine learning, to achieve in both cases human-like behaviour and strategies, in Quake™ 2 and 3 games respectively. Recent studies related to computational intelligence (a branch of the AI) , are based on bots controlled by neural networks (NNs). For instance, the authors in [15] train NNs by reinforcement learning, and Schrum et al. [16] evolve NNs searching for multimodal behaviour in bots.

The design of human-like bots (try to imitate humans' behaviour) is an interesting research line which has become popular a few years ago in the FPS scope, and specifically inside UT2K4 series due to the international Botprize competition [17], which searches for the 'most human' bot. Several papers have been published regarding this area, such as the work by Soni and Hingston [6], in which the authors use a NN to train a bot, based in recorded data of human plays, for playing UT2K4 as a human. Schrum et al. [18] applied a similar strategy, relying in human records of matches in the same game to support the multiobjective evolution of a NN. This is evolved to get the best performance and skilled actions, but the bot humanity is corrected, mainly regarding the navigation capability, by means of the human data. There are other approaches such as [19] which is based in reinforcement learning and self-organizing NN, or [20] in which the authors model a part of the brain workspace and neural interactions (through NN).

Our work is enclosed in this scope, however it presents the design of a human-like bot created by modelling its AI relying in the knowledge of a human expert player. It presents a novel state-based approach, which considers main and sec-

ondary states, and it also deeply describes the AI engine workflow, and the realistic (from the human point of view) use of a database for 'learning' the arenas.

4 UT2K4 Expert Bot

The design of the expert bot (*E-Bot*) AI is based on the knowledge of an expert (human) player, *Me$$!@h*, who belongs to the best Spanish UT2K4 clan, *Trauma Reactor* (*dRâ*), and has participated in some European championships. He is one of the authors of the paper and has designed and implemented the main work flow of the bot's behaviour. As a summary, *E-Bot* is implemented considering hierarchical states (primary and secondary), a set of rules to decide the correspondent states, and a (simple) database. In the following sections all these terms will be described, starting with a summarised expert analysis of the main elements in the game.

It is important to point out that we have considered the rules of the official championships then, as stated, weapons are not respawned, there are some forbidden items, and there is a time limit per match instead a number of frags limit.

4.1 Expert knowledge

The items and weapons are critically valuable in a *1 vs 1 Death Match*, since they could mean the difference between win or lose against an opponent. It is even more important to consider the respawn *timing* of each of them, i.e. the time it takes to appear again once an item has been picked (weapons are not respawned). The importance grows in this mode since if one player picks one item or weapon, she prevents the opponent for profiting it (while she does, of course). Expert players must control this time in order to move to a known respawn area in the moment an item appears. These conditions (timing and locations) depend on the map, which the player should know in advance.

The following list describes the main *items* to consider:

- *Super Shield*: the most important item in 1 vs 1 matches. It gives the player 100 shield points (the maximum is 150).
- *Small Shield*: similar to previous one but it gives 50 shield points.
- *Health Pack*: gives 25 health points. It just can be used if the player's health is lower than 100.
- *Health Vial*: gives 5 health points. This can be always used until the player reaches 199 health points.
- *Adrenaline*: gives 5 adrenaline points. If the player reaches 100 of these points, she can obtain a reward in a limited time, such as:
  - Invisibility: the player cannot be seen.
  - Berserker: faster shots.
  - Speed: faster movement.

- Booster: player's health and shield is regenerated.

– *Ammo*: gives ammunition for a specific weapon. There is a different item per weapon.

Weapons in UT2K4 have been 'carefully' designed to be all of them equally useful and important (as a difference to other games), so there is no one absolutely better than the rest. Every weapon has a perfect moment to be used and take advantage over other weapons. The choice of this moment is an expert decision that depends on several factors such as the player's and enemy's health, the distance to enemy or the height/level where she is placed. The list of weapons along with their utility is summarised in Table 1.

Table 1. UT2K4 Weapons.

	Useful for/when	*Main shot*	*Secondary shot*
Shield Gun	retreat and protection, player's health low	can damage enemy, impulse for jump	shield for reducing the received damage
Assault Rifle	hidden enemy known	fire shot, low damage	parabolic launch of grenade
Link Gun	enemy tries to escape, covering an area	energy balls in straight direction	links the enemy to the ground
Bio Rifle	surprising the enemy, enemy's health is higher than ours	viscous balls which glue to floor and walls, lightly damage to enemy who moves close to them	charge, big ball which could kill the enemy if he is close
Minigun	enemy's health is low, medium/close distance	very fast small bullets	slow big bullets
Rocket Launcher	medium distance	one rocket, high damage on impact	several rockets, spam shots
Flak Cannon	enemy is on different level	burst of bullets, close distance	parabolic ball
Shock Rifle	all distances, very versatile	energy laser	energy ball which can explode, high damage
Lightning Gun	far distances	sniper rifle	sniper sight

The *movement* is another key factor in UT2K4, since it could mean the difference between win or lose. Players usually move jumping, in order to avoid being shot easily. This is a problem of our approach because the Pogamut movement module does not implement the jump.

4.2 Database

When a human player starts playing in a map unknown to him, it is usual to take some turns to investigate/analyse the scenario and 'learn' where the important things are, i.e. mainly weapons and items, but also advantageous positions and hiding places.

This behaviour has been implemented in the expert bot by means of a simple database. It features a single table, which has as fields:

– *Unreal_ID*: unique identification of an item in Pogamut.
– *Type*: health, shield, ammo, or weapon, among others.
– *Name*: for instance Flak Cannon, Small Shield or Mini Health.
– *Map*: name of the map where the item is placed.

The initial idea was to include lots of information, such as respawn points, usual paths, heat zones, last plays or moves, which will imply a high level of knowledge for the bot, but also a high computational cost when the it would have to interpret and use all this data. Since the bot must react in real-time, we decided (for this initial version) to store just the most important and useful information. To this end we performed several test matches using different combinations of these data. Finally we concluded that the most relevant data to record and recover later were the described above.

The most important field is *Unreal_ID*, because Pogamut offers information about it, including its location. It is important to note that the table is filled with the information about the items that the bot discovers while it is moving across the map (searching for enemies or just exploring to learn). There are no records about items not being seen by the bot, although it could be possible in Pogamut (this would be a trick).

This database has been designed and implemented using SQLite, due to its light 'weight', simplicity and portability. It just consists in a file which we can move together with the bot code from one PC to another one. It will be extended in future versions.

4.3 AI work flow

The use of Pogamut implies we cannot consider the FSM defined in the UT2K4 bot's AI, thus, a complete AI engine must be designed and implemented. Pogamut offers some high-level functions which we have profited, such as basic navigation from point to point in a map. As commented in the introduction section, Pogamut offers to the user sensory information that the bot perceives during the match. To this end the bot can implement the so-called *listeners*, which are triggers that the programmer defines in advance; when an event related to this trigger happens, the bot performs the associated actions.

The main idea in *E-Bot* AI module is the consideration of two levels of states: primary and secondary. The first ones correspond to general actions to perform, while the latter are devoted to perform additional tasks within the main action flow. For instance, the bot can decide to attack the enemy, but if its health is low also search for any health item.

The *primary states* descriptions are:

- *Attack*: the enemy is in sight. The bot moves towards him, shooting with its best weapon and avoiding opponent's shots. Otherwise, the bot moves side to side (in a pendular movement).
- *Hunt*: the bot tries to hunt the enemy. If he is in sight, it pursues him; otherwise the bot infers its position considering the noise that the enemy does.
- *Retreat*: the bot tries to escape from the enemy. If he is in sight, the bot moves facing him, avoiding his shots and using the shield weapon (if possible). Otherwise the bot shoots at him while it escapes. If the enemy's position is not known, the bot infers it hearing at the noises he produces and moves far from them.

– *Greedy*: the bot moves around the map picking the items it needs. The bot knows where they are if it has been previously in the map and has stored its position in the database. If the enemy is in sight, the bot tries to defend itself while it picks the items.
– *Camp*: the bot waits for the enemy in a static position.

The *secondary states* are only devoted to change the movement of the bot:

– *Defensive Profile*: the bot tries to escape from the enemy.
– *Offensive Profile*: the bot moves towards the enemy.
– *Pickup Weapon*: the bot picks all the weapons it founds.
– *Pickup Ammo*: the bot picks all the ammunition it founds.
– *Pickup Health*: the bot picks all the health items it founds.
– *Critical Health*: the bot moves towards the closest health item it knows (using the database).
– *Critical Weaponry*: the bot moves towards the closest weapon it knows (using the database).

The choice of the primary state is performed by the AI engine following the flow chart shown in Figure 1. The *Camp* state is not included because it is

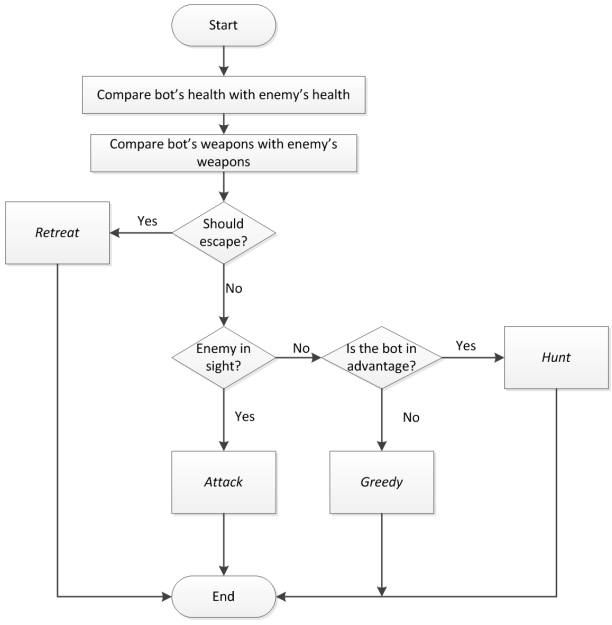

Fig. 1. Primary States selection flow chart

a special state with very particular conditions. The comparisons and decisions required to choose the state are quite complex, because they consider several factors and situations that the human expert knows. For instance, the comparison between weapons depends on several parameters and weights, because of the

existent balance between the power and usefulness of all of them, which could mean that a weapon is the best in a specific situation (different levels, hidden opponent, close or far enemy), but it could be the worse on a different position.

The AI engine (composed by a huge system of rules) chooses the primary state at a time, but while the bot is performing the actions associated to it, the engine continues checking conditions (for instance the timing of every item), receiving sensory information, or checking the bot status, for instance. This way, a secondary state can also be set. However the engine can stop and change, at any time, both the primary and/or the secondary state, depending on the conditions, received information and status, giving an extra 'flexibility' level which a FSM usually do not offer.

As a general summary, the bot tends to be defensive if its health is lower than the enemy's, unless the latter has a critical health level, so the bot will attack him to win a frag. In similar health conditions the attitude depends on the weaponry (in comparison with the enemy's weapons). If the bot's health is good, it is quite aggressive. However, there are several conditions and situations considered which could change the states and actions to perform. For example, the difference in height (levels). The *E-Bot* general flow chart can be seen in Figure 2.

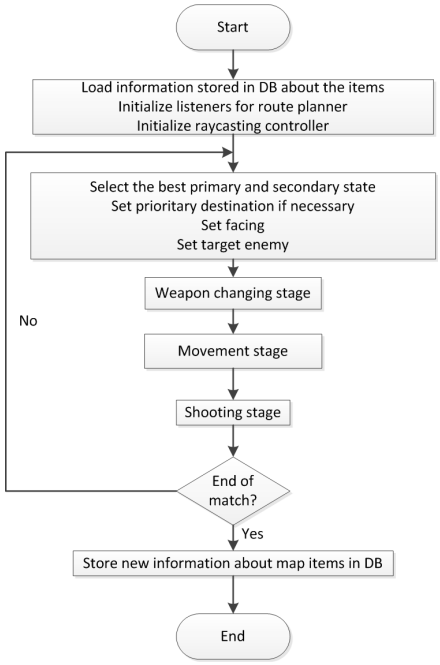

Fig. 2. *E-Bot* general flow chart. The route planner controls the navigation around the map, while the raycasting controller traces vision lines for avoiding obstacles and seeing enemies.

The source code of *E-Bot* is available under a GPL license at `https://github.com/franaisa/ExpertAgent`.

5 *E-Bot* Results

In order to test the expert bot some experiments has been conducted. The first consists in launching four different game matches in the *1 vs 1 - Death Match* mode, having as players a game standard bot in the hardest difficulty (*StdBot*), and our expert bot (*E-Bot*).

Two different arenas have been considered, namely *DM-Idoma* and *DM-Ironic*, since they are two of the most famous maps, used in competitions and with a design that offers almost any possible fighting situation: different levels, hidden zones, viewpoints and camping areas, etc.

The bots have been fighting following the commented championship rules (as *E-Bot* was designed) along 15 minutes. The results of every match as well as the average in each map are shown in Table 2.

Table 2. Scores (number of frags) of the test fights between *E-Bot* and a Standard UT2K4 bot in the hardest difficulty (*StdBot*).

Map	*E-Bot*	*StdBot*
DM-Ironic	17	6
	22	8
	19	9
Average	19.33	7.67

Map	*E-Bot*	*StdBot*
DM-Idoma	23	14
	20	9
	25	8
Average	22.67	10.33

As it can be seen *E-Bot* outperforms the standard UT2K4 bot in all the matches, even in the hardest difficulty level, which is a true challenge for a medium level human player. This leads us to conclude that the expert bot has been successfully designed and implemented.

As an additional test, our expert human player fought some matches against *E-Bot*, always beating it. It is an expected result, since this player is one of the best in Spain. Thus, we have also proved the value of the expert bot versus four medium level human players (they play frequently but they are 'non-professional'). One of them lost all the matches against the bot, and the rest had serious difficulties to beat *E-Bot*, yielding a considerable amount of frags in the matches, so it could be considered as high-level enemy.

6 Conclusions and Future Work

In this work, the design of a human-like bot for the First Person Shooter game Unreal Tournament™ 2004 (UT2K4) has been described. It models the expert knowledge of a Spanish high-level player so, its has been named Expert-Bot (*E-Bot*). Its behaviour is shapes as a finite state machine with two levels of states (primary and secondary), each of them with an associated priority for performing actions. This work flow is flexible, so the AI engine can alter or change

it depending on the conditions and on the bot's or enemy's status. Moreover, the bot uses a database for 'learning' the maps, as a human player would do the first time she navigates around a new arena. *E-bot* has been designed for fighting in 1 vs 1 death match mode, considering the official competition rules.

Some experiments have been conducted, firstly proving that *E-Bot* outperforms the standard UT2K4 bots, even in the hardest difficulty level. Then, a set of matches against medium-level human players have been performed with quite good results: one human has been beaten and the rest have had high difficulties (several frags) to win. The authors consider these results as a success, however there still remain several points of improvement to create a high competitive bot. According an expert player's opinion (one of the authors), *E-Bot* does not perform as well as expected (from the behavioural point of view), so it would be necessary to improve its behaviour rather than its performance in results.

This way, the first line of research could be the improvement of the bot in its main flaw, the movement, since the implementation which offers Pogamut does not include jumps, which in fact are basic in an expert control. Thus, the possibility of jumping must be included in its actions, in order to model a harder opponent (more difficult to kill at least).

The database complexity could be increased for considering other important information for the bots, such as respawn points, heat zones, better paths, navigation points. But some additional work should be done in order to deal with this data in real-time, avoiding a delay in the performance of the bots.

Once the behaviour would be improved, then several researching could be made with regard to the finite state machines improvement (set of behavioural states). Following other approaches by the authors, evolutionary computation algorithms [3, 8] could be used to improve both, the set of behavioural rules and the set of threshold parameters which, in term, determine which rules are triggered and so, define the bot's behaviour.

Finally, another research line could be the adaptation or optimisation of *E-Bot* for multiplayer and team modes, which are so far the most famous among the players.

Acknowledgements

This paper has been funded in part by projects P08-TIC-03903 (Andalusian Regional Government), TIN2011-28627-C04-02 (Spanish Ministry of Science and Innovation), and project 83 (CANUBE) awarded by the CEI-BioTIC UGR.

References

1. WEB: (Gamebots project) http://gamebots.sourceforge.net/.
2. WEB: (Pogamut 3 project) http://pogamut.cuni.cz/main/tiki-index.php.
3. Montoya, R., Mora, A., Merelo, J.J.: Evolución nativa de personajes de juegos de ordenador. In et al., E.A., ed.: Actas primer congreso español algoritmos evolutivos, AEB'02, Universidad de Extremadura (2002) 212–219

4. Jacobs, S., Ferrein, A., Lakemeyer, G.: Controlling unreal tournament 2004 bots with the logic-based action language golog. In: Proceedings of the First Artificial Intelligence and Interactive Digital Entertainment Conference. (2005)
5. Karpov, I., D'Silva, T., Varrichio, C., Stanley, K.O., Miikkulainen, R.: Integration and evaluation of exploration-based learning in games. In Louis, S.J., Kendall, G., eds.: CIG, IEEE (2006) 39–44
6. Soni, B., Hingston, P.: Bots trained to play like a human are more fun. In: IEEE International Joint Conference on Neural Networks, IJCNN'08. (2008) 363–369
7. Small, R., Bates-Congdon, C.: Agent Smith: Towards an evolutionary rule-based agent for interactive dynamic games. In: IEEE Congress on Evolutionary Computation 2009 (CEC'09). (2009) 660–666
8. Mora, A.M., Montoya, R., Merelo, J.J., García-Sánchez, P., Castillo, P.A., Laredo, J.L.J., Martínez, A., Esparcia, A.I.: Evolving bot AI in Unreal. In et al., C.D.C., ed.: Applications of Evolutionary Computing, Part I. Volume 6024 of LNCS., Springer (2010) 170–179
9. Esparcia-Alcázar, A.I., Martínez-García, A., Mora, A.M., Merelo, J.J., García-Sánchez, P.: Genetic evolution of fuzzy finite state machines to control bots in a first-person shooter game. In Pelikan, M., Branke, J., eds.: GECCO, ACM (2010) 829–830
10. Booth, T.L.: Sequential Machines and Automata Theory. 1st edn. John Wiley and Sons, Inc., New York (1967)
11. Cole, N., Louis, S.J., Miles, C.: Using a genetic algorithm to tune first-person shooter bots. In: Proceedings of the IEEE Congress on Evolutionary Computation 2004. (2004) 139–145
12. Priesterjahn, S., Kramer, O., Weimer, A., Goebels, A.: Evolution of human-competitive agents in modern computer games. In: IEEE World Congress on Computational Intelligence 2006 (WCCI'06). (2006) 777–784
13. Thurau, C., Bauckhage, C., Sagerer, G.: Combining self organizing maps and multilayer perceptrons to learn bot-behaviour for a commercial game. In Mehdi, Q.H., Gough, N.E., Natkine, S., eds.: GAME-ON, EUROSIS (2003) 119–123
14. Zanetti, S., Rhalibi, A.E.: Machine learning techniques for FPS in Q3. In: ACE '04: Proceedings of the 2004 ACM SIGCHI International Conference on Advances in computer entertainment technology, New York, NY, USA, ACM (2004) 239–244
15. Cho, B.H., Jung, S.H., Seong, Y.R., Oh, H.R.: Exploiting intelligence in fighting action games using neural networks. IEICE - Trans. Inf. Syst. **E89-D**(3) (2006) 1249–1256
16. Schrum, J., Miikkulainen, R.: Evolving multi-modal behavior in NPCs. In: Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium On. (2009) 325–332
17. 2K-Games: The 2k botprize competition (2012) http://www.botprize.org.
18. Schrum, J., Karpov, I.V., Miikkulainen, R.: Ut$\hat{2}$: Human-like behavior via neuroevolution of combat behavior and replay of human traces. In: Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG 2011), Seoul, South Korea, IEEE (2011) 329–336
19. Wang, D., Subagdja, B., Tan, A.H., Ng, G.W.: Creating human-like autonomous players in real-time first person shooter computer games. In: Proceedings of the 21st Annual Conference on Innovative Applications of Artificial Intelligence (IAAI'09), Pasadena, USA (2009) 173–178
20. Fountas, Z., Gamez, D., Fidjeland, A.: A neuronal global workspace for human-like control of a computer game character. In Cho, S.B., Lucas, S.M., Hingston, P., eds.: CIG, IEEE (2011) 350–357