Learning behaviours for bots in first-person shooter games

Rowan Liddell,
David C. Moffat
Department of Computing
Glasgow Caledonian University, UK.
D.C.Moffat@gcu.ac.uk

Abstract.

First Person Shooter (FPS) games are a popular genre in the games industry, yet AI routines used for AI controlled enemies or 'bots' often rely on techniques that allow little to no adaptation to different players' play styles and that do not scale well as the complexity of the game world increases. While previous AI research has shown that more believable bot AI can be created through a variety of techniques, much of the bot learning research has been performed offline in training scenarios rather than in real time as the game is being played.

A bot was developed to play matches in Unreal Tournament 2004, using the Pogamut middleware. The project saw the development of a decision tree structure to govern the bot's high level decision making processes. Memory structures were defined to allow the bot to keep track of its experiences which, together with environmental information, provided the knowledge basis for the bot's decision making. A reinforcement learning algorithm was used to provide feedback to the bot based on the outcome of its decisions. The AI code was used in two bots - one with an active learning mechanism, the other identical in all aspects save a learning rate of zero - which were then tested by playing against AI controlled opponents in a deathmatch scenario.

Results showed that the bot with learning demonstrated evidence of planning and of reaction to the environment and developed characteristic behaviour traits, thereby establishing proof of concept. Comparison with a non-learning bot, and performance analysis of the bot using different learning rates, indicated that the reinforcement feedback mechanism played an important part. There was little performance cost, and the bot learned to outperform the standard bots that come with the game.

1 Introduction to FPS bots

First-person shooter (FPS) games are a popular genre of computer games. They have players shooting other players in the virtual world, seen in the screen as a first-person perspective. When other players are not available, or not enough of them to make enough enemies, the computer can play bots as AI characters.

As far back as 1999 Woodcock noted that game AI was assuming greater importance in game development [13]. More powerful hardware, even at that time, allowed more resources to be devoted to AI — although Woodcock also acknowledges that players expect high standards of graphics which have traditionally consumed most of the available hardware resources.

However, while advances in graphics have been astonishing since 1999, AI has not seen a similar progress in development. Graphics still consume the vast majority of CPU cycles, and remain a selling point. Yet while the need for better AI is evident in many reviews, techniques used for the non-player character (NPC) opponents or bots have not progressed as much as graphics or physics simulations, often leading to NPC behaviour which is predictable and unrealistic [11].

Reasons for the lag in AI development may include the cost of AI algorithms, and another may be the difficulty of deriving effective ones. Bots in FPS games still depend on relatively simple program schemes, such as finite state machines, or some variant of them. While convenient to program, and fast to execute, these techniques can result in predictable behaviour, that does not adapt to the player.

Adaptive learning is a technique proposed by Bakkes et al [1], where the AI agent uses existing knowledge of the world available to it to develop a strategy in real time, allowing it to adapt to continuously changing situations. They applied their technique to a bot in a Real-Time Strategy (RTS) game, and concluded that given appropriate initial knowledge the AI is able to adapt.

This paper reports an attempt to apply simple learning techniques to FPS bots, to make them able to adapt to different opponents, and learn how to beat them. A new bot is designed, based on existing technology, for the game UT2004, but with the addition of a reinforcement learning component to modify its decision tree architecture. Whether the learning algorithm can learn quickly enough to adapt to the opponent in a reasonable time is one question; and performance is another. If the new algorithms are also quick to execute, they might form the basis of a new generation of FPS bots.

Unreal Tournament 2004 (also referred to as UT2004) is a first-person shooter game made by Epic Games. The game has native bots of various difficulty settings which provide opponents for players when no other human players are available. Unreal Tournament 2004 has been used in a number of research projects, such as those carried out by van Hoorn [12] and by Esparcia-Alcazar [3].

What makes UT2004 outstanding as a research environment is the availability of APIs and IDE plug ins through the Gamebots and Pogamut projects, which facilitates the development of AI-controlled agents to a great degree. Gamebots as an AI research framework was first proposed by Addobati et al in 2001 [8] who set out to provide a research infrastructure which combined an FPS game environment and its dynamic characteristics with a module that allows AI agents to be connected to a server and controlled via network messages. En-

vironmental information is sent from the Gamebots server to the bot, while the bot's actions are sent back.

The Gamebots framework was developed further [2], and used with UT2004 to provide the arena for contestants in the 2008 Bot Prize competition [6]. The software framework was developed into Pogamut 3, which software provides a Java library, integration with the Netbeans IDE and a number of data gathering and visualisation tools [5]. Various sample bots are available, one of which, 'Hunter' was used for some of the bot opponents. The Pogamut agent is created in the Netbeans IDE and connects to the UT2004 server using the network protocol provided by the GameBots 2004 API.

Although the game UT2004 is ten years old now, we used it as our testbed because the Pogamut system was built to allow bot AI to be developed experimentally with that game. This makes such research easier to do, but it also ensures a higher degree of realism, because the Pogamut approach means that bots are client-side to the game, and so do not have easy access to information in the game world that give them an unfair advantage over human players.

2 A bot that learns

The bot is programmed using the Pogamut system, which gives it access to the same basic library functions that the other bots in Pogamut and in UT2004 all have.

Above those functions, the structure of our bot is coded as a decision tree, with three different levels. It has an upper architecture shown in Figure 1, which incorporates a degree of learning from experience. The three levels are where decisions are taken to select firstly which emotion or mood (select behaviour); then which goal to plan for, such as to attack or run away (set goal); and finally which action to take, such as to choose a weapon (take action). These three levels might also be thought of as levels for strategy, tactics and action.

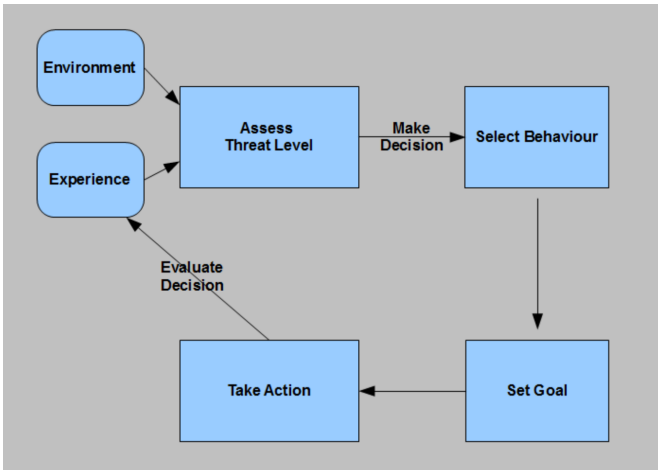

Figure 1. Architecture of the agent

At each level, the decisions are taken depending on thresholds for key variables, and it is these thresholds that can change with experience. The most important variable is the bot's own threat-level, which expresses how threatening it is to other bots. It is a function of several basic variables (equation 1), that are available from the Pogamut programming layer, including the bot's own health h, its current armour a, and its own history of kills and deaths.

$$threat = h/100 \; + \; a/150 \; + \; (ka - da)/25 \; + \; (ke - de)/25 \quad (1)$$

The kills are the total kills of all other bots ka added to those of the current enemy bot ke. Likewise the deaths caused by any other bot. da, and by the enemy bot de. The factors are normalised so that they are at comparable levels, and can each reach a maximum of one. The maximum health in the game is 110, the maximum armour in most circumstances is 150, and a match is complete when the first bot reaches 25 kills, which is where the numbers n the equation come from.

This threat level determines the behaviour (mood) of the bot, which is the first decision the bot makes, at its top level. The mood corresponds to the branch taken at the top of the decision tree, and the lower tree below that correspond to goals and actions selected.

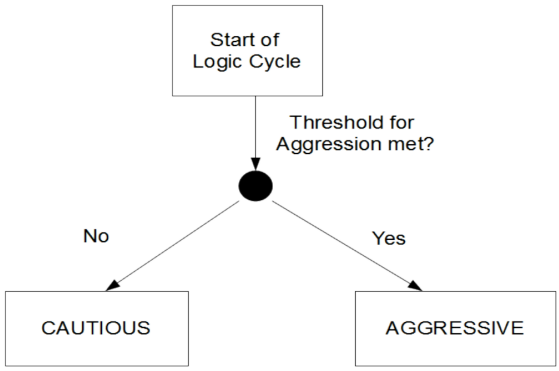

Figure 2. Top-level behaviour selection of mood

The goals associated with these two behaviours are to score a kill (if aggressive), or to improve strength (if cautious). See Table 1. They will be used to evaluate the effectiveness of the bot's decisions, and thus learn to make better decisions in future.

Table 1. The bot's desired outcomes for each behaviour

Behaviour	Desired outcome	Goal
aggressive	score a kill	KILL
cautious	improve own threat level	STRENGTHEN

When it is cautious, then, the bot is more likely to withdraw, in order to find health kits or ammunition pickups, to build up its strength before it fights later on. When the bot is aggressive, it is more likely in the next decision, for tactics, to go on the offensive. It will not try to improve its own strength, but will start to hunt for an enemy to fight, as shown in 3.

At this level of tactical planning, there are more goals available, as shown in Table 2

The decision tree continues to lower layers where things like weapon choice are decided, or different methods to improve strength, whether that be to search for health or armour or ammunition, which

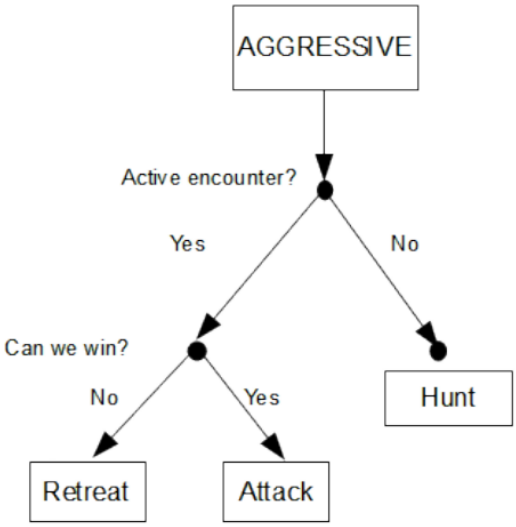

Figure 3. Top-level behaviour selection of mood

Table 2. The bot's desired outcomes for each plan

Behaviour	Desired outcome	Goal
attack	score a kill	KILL
retreat	avoid being killed	NOT DIE
hunt	none – form new plan when target found	
improve	improve own threat level	STRENGTHEN

may all be found at appropriate locations. There are more than thirty nodes in the entire decision tree.

When there is an outcome to the planning cycle, the possible values returned (see Table 3) are used in the learning algorithm.

Table 3. The outcome values for goals

Outcome	Description	Value
KILL	bot scores a kill	1
STRENGTHEN	own threat level improved	0.5
NONE	No significant change to threat level	0
WEAKEN	own threat level worsened	- 0.5
DIE	bot is killed	-1

These reward valued then modify the decision thresholds, with a separate learning rate for each level of the decision tree. For example, if the bot had selected the **aggressive** behaviour, wanting to kill the enemy, and it then succeeded in doing so, then its desired outcome value (of $D = 1$) would exactly match the outcome achieved ($A = 1$), giving a net error of zero ($E = A - D = 0$). On the other hand, if it got killed itself, then the error ($E = -2$) would cause the behaviour threshold to decrease, as in equation 2.

$$aggressive' = aggressive + \alpha B \times E \quad (2)$$

The learning rate for behaviours is αB, which might be set to 0.1 for the bot, say. When the bot starts, it may have a threshold of 0.5, so if the first thing it tries to do is to kill the enemy, but it fails and is itself killed, then the new threshold for aggression will be $0.5 + 0.1 \times -2 = 0.3$, and so the bot will be significantly more cautious in future.

There are separate learning rates for the planning decisions, and for the actions chosen (αP and αA respectively). In pilot testing of the bot, against a version of itself that did not learn at all (with its learning rates all set to zero), a good set of value for the three learning rates was found to be $\alpha B = \alpha P = 0.2$ and $\alpha A = 0.15$. These were the values used for the bot in the experiments to evaluate its performance.

3 Experiment

Both the UT2004 server and the Netbeans project connecting the bot client were run on the same computer. The AI bot was observed and developed using two existing game levels or 'maps' of UT2004 designed for the deathmatch game mode. In this game mode, a number of players fight against each other in order to gain the highest number of kills. A kill is scored when an opponent's health is reduced to 0, whereupon the opponent respawns. This game mode is fast paced and provides a large number of encounter opportunities for the bot to gain experience.

Performance analysis was done by comparing the bot's kill and death statistics with those of other players. The bot's evaluations of its own threat level was one variable logged against the encounter index. An encounter is defined as a battle from the time when the bot meets its opponent to the point when one of them dies. Decision outcome was also logged against the encounter index to show how decision accuracy developed as the game progressed, with the intention of determining whether the modification of the learning algorithm in response to feedback produced the desired result.

As this project was intended as a proof of concept, it was expected that while results will show a conclusive change in bot behaviour, these changes may only become evident through statistical evaluation.

The map chosen for the evaluation trials was DM-DE-OSIRIS2, a large map with an ancient Egyptian theme. This map features a large number of pick-ups, a complex layout, and few lifts where the pathfinding algorithm might encounter problems, and no damage inflicting environments.

In order to test the AI and gather data for evaluation, two instances of the bot are spawned in the selected map – "Dec" is the learning bot, with non-zero learning rates, and "Cas" is an identical bot except that its learning rates are set to zero. The bots are thus both the same to start with, but as they play, one of them learns from its experience and should change its behaviour.

Data are gathered for both bots in order to determine any differences in behaviour caused by the learning algorithm. The map is then further populated with a selection of native UT2004 of varying levels (e.g. 'godlike' and 'novice') as well as instances of the 'Hunter' Pogamut bot. The range of bot opponents allowed a comparison of Dec the bot's performance against other standard bots, as well as against itself in the non-learning form of Cas. The deathmatch tournament was run with all the bots in the level, until both Dec and Cas had experience at least a hundred encounters (kills or deaths).

4 Results

By the end of the tournament, Dec had had 102 encounters, and Cas 104. Some of these were "default" encounters meaning that the other bot was not selected as a target to fight, because there were multiple other bots and only one at a time can be a target. The remaining

majority of encounters were either aggressive or cautious ones, depending on the behaviour that the bot chose to adopt. The two bots Dec and Cas were clearly different, as Dec was aggressive 19 times, and cautious much more often, at 69 times; but Cas was aggressive 46 times, and cautious only 28 times. It appears that Dec must have learned that caution was wise in this environment, while Cas persisted in rash aggression.

This appearance is borne out in the tournament results that show which bots killed more, or died the most. See Figure 4

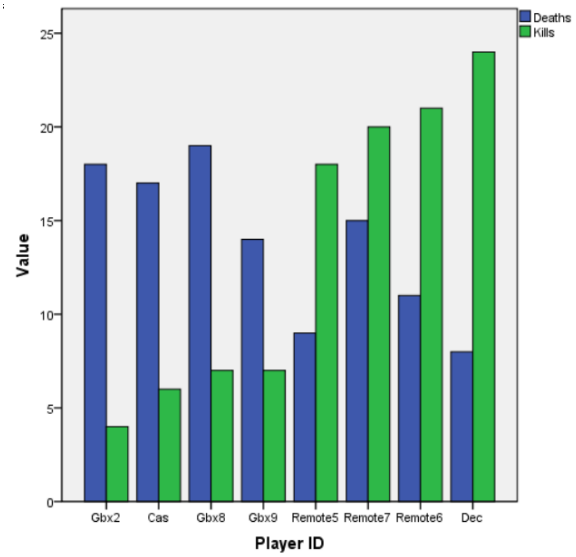

Figure 4. Kills and deaths per play: worst performing bots at left

The standard bots are name GbxN, which are the ones that come with UT2004. Stronger are the bots from the Pogamut software framework, called RemoteM, and they kill more often than the GbxN bots, and do not die as much. Cas and Dec are greatly different from each other in effectiveness, with Dec learning how to kill many more bots (24 against 6), and dies on fewer occasions (7 against 17). This is a clear result in favour of the effectiveness of the learning algorithm, which Dec has but Cas does not.

It is also interesting to see the performance against the other bots in the tournament. Cas performs at a level comparable to the standard UT2004 bots, which are the weaker ones in the tournament. Dec performs as the Pogamut bots, on the ohter hand, which are the strongest. In fact Dec is by a small margin the strongest performer of all.

The bot therefore has achieved a good performance. It remains to ask how it does this; and whether the computational cost is acceptable.

The logs of the tournament show howthe key bot variables change, and one of them is for the *engagement threshold*. This threshold determines the goal for the bot when it is being attacked: whether it will fight or retreat depends on its health value being above the threshold. The threshold is fixed at 50 for the bot Cas, and so it starts there as well for bot Dec. As he tournament progresses though, Dec's threshold increases steadily, under the influence of the learning process. by the end of the tournament it is over 56, meanint that it has put on 12% of its original value, making it less likely that Dec will engage with enemies, unless its own health is high. This would have led to Dec breaking off combat engagements earlier, and so increasing his chances of surviving a fight, as indeed the data seems to support. The difference in Dec's success can thus be accounted for by learning to raise this engagement threshold, and presumably by the other decision thresholds also changing.

Finally, the frame-rate of the system was recorded when the bots were running as Dec or Cas in a tournament as above. It was then recorded again with exactly the same conditions, except that both Dec and Cas were substituted by two native GbxN bots. The AI bots did cause a slight slow-down of the frame-rate, but it was only about 0.5%, from 90.49 frames per second, to 90.08 frames per second.

5 Related work

Other researchers have used the same software framework to develop new AI for bots in the game UT2004. The chief efforts have been done as part of the annual Botprize competition for video game AI [6, 7].

Notable work in this competition includes the neural network training for bots' weapon selection [10]. The training algorithm was back-propagation, and the bots were able to perform at a higher level because of learning to make better choices of weapon.

Decision trees were used by Fernández Leiva et al [4], to drive the bots' decisions, and they were evolved to make the quality of decisions improve over time. The aim of this research was a little different than our own approach, in that the bots were intended to learn to behave in ways that would be more enjoyable for the player.

Patel et al [9] evolved more intelligent bots by using a different machine learning technique: Q-learning. They applied this to make the bots learn how to fight better. Q-learning is typically a simpler and more efficient algorithm than neural nets with back-propagation, and yet the bots' performance was still improved by it. This was partly because Patel et al limited the operation of the learning algorithm to small parts of the bots' behaviour, rather than letting it operate over the whole behaviour repertoire in one scope. They chose Q-learning over reinforcement learning because of a concern that reinforcement learning would be overwhelmed by the range of possibilities.

Our research reported here was intended to improve the performance of the bots in the game, in terms of score, survival and winning. These other researchers also took the view, as did we, that the commercial bots in the game were conventionally programmed as specially crafted code directly written by programmers themselves. They investigated ways to help automate the process by giving the bots some kind of learning. Our own approach has been similar, in that we limited the scope of learning to three small points (including weapon selection), as layers of a decision tree; but we did use a simple reinforcement learning scheme. However we found that the learning was not overwhelmed, and the bots were able to perform at a fairly high level, with only a small cost in computation time.

6 Conclusion

AI bots in FPS games can be made to learn, and as the experiment shows their performance can then improve significantly. The learning bot here became the best performer, compared with native UT2004 bots and with other Pogamut bots that were specially coded. This comparison should be cautious, however. UT2004 bots are intended to be fun to play against, and not necessarily to be high performers.

Nevertheless it seems that it is quite feasible to make even the standard FPS style of bot or non-player character perform at a high level,

or otherwise adjust its play to the circumstances of the level, and especially of other players. The performance of the learning scheme, using reinforcement learning over a decision tree, was good, costing only a little extra computation time, and allowing the game to maintain almost the same frame-rate as it achieved with the learning algorithm turned off. However this was only a comparison between the algorithm with and without learning, and not a fuller evaluation that compared the bots with the other bots in the game. Such a comparison could need a more involved evaluation scheme because the bots would behave in quite different ways.

It is possible that similar techniques as ours could be used to make a bot adapt to players in ways that make for a fun opponent rather than an overpowering one; although we did not attempt to do that.

The techniques were not very difficult to program, using a decision tree together with a learning function that adapts its threshold values, simultaneously at multiple layers. This is a generalisable technique, and suggests that the evolution of FPS bots is not over yet. The code does not take a lot of computer power either, which is often a problem with AI in games. We conclude that giving learning algorithms for FPS bots, and probably other kinds of AI non-player character in games, is a promising line of research.

7 Acknowledgements

We are grateful to the members of the Pogamut team for making Pogamut 3 such an ideal environment for research in bot AI, and for the active support offered by its developers via their forums.

REFERENCES

[1] S. Bakkes, P. Spronck, and J. van den Herik, 'Rapid and reliable adaptation of video game ai', *IEEE Transactions on Computational Intelligence and AI In Games*, **1**(2), 93–104, (2009).

[2] M. Bida, M. Cerny, J. Gemrot, and C. Brom, *ICEC 2012*, volume 7522 of *LNCS*, chapter Evolution of GameBots project, 397–400, Springer, Heidelberg, 2012.

[3] Anna I. Esparcia-Alcazar, Anais Martinez-Garcia, Antonio Mora, J J Merelo, and Pablo Garcia-Sanchez, 'Controlling bots in a first person shooter game using genetic algorithms', in *IEEE Congress on Evolutionary Computation*, p. nil, (7 2010).

[4] Antonio J. Fernández Leiva and Jorge L. O'Valle Barragán, *Foundations on Natural and Artificial Computation: 4th International Work-Conference on the Interplay Between Natural and Artificial Computation, IWINAC 2011, La Palma, Canary Islands, Spain, May 30 - June 3, 2011. Proceedings, Part I*, chapter Decision Tree-Based Algorithms for Implementing Bot AI in UT2004, 383–392, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[5] Jakub Gemrot, Rudolf Kadlec, Michal Bída, Ondřej Burkert, Radek Píbil, Jan Havlíček, Lukáš Zemčák, Juraj Simlovič, Radim Vansa, Michal Stolba, Tomáš Plch, and Cyril Brom, *Pogamut 3 Can Assist Developers in Building AI (Not Only) for Their Videogame Agents*, 1–15, Lecture Notes in Computer Science, Springer Science + Business Media, 2009.

[6] Philip Hingston, 'A turing test for computer game bots', *IEEE Transactions on Computational Intelligence and AI In Games*, **1**(3), 169–186, (September 2009).

[7] *Believable Bots*, ed., Philip F. Hingston, Natural Computing, Springer Science, 2012.

[8] Gal A. Kaminka, Manuela M. Veloso, Steve Schaffer, Chris Sollitto, Rogelio Adobbati, Andrew N. Marshall, Andrew Scholer, and Sheila Tejada, 'Gamebots: a flexible test bed for multiagent team research', *Communications of the ACM*, **45**(1), nil, (2002).

[9] P. G. Patel, N. Carver, and S. Rahimi, 'Tuning computer gaming agents using q-learning', in *Computer Science and Information Systems (FedCSIS), 2011 Federated Conference on*, pp. 581–588, (Sept 2011).

[10] Stelios Petrakis and Anastasios Tefas, 'Neural networks training for weapon selection in first-person shooter games', in *Proceedings of the 20th International Conference on Artificial Neural Networks: Part III*, ICANN'10, pp. 417–422, Berlin, Heidelberg, (2010). Springer-Verlag.

[11] Yingying She and Peter Grogono, 'An approach of real-time team behavior control in games', in *2009 21st IEEE International Conference on Tools with Artificial Intelligence*, p. nil, (11 2009).

[12] Niels van Hoorn, Julian Togelius, and Jurgen Schmidhuber, 'Hierarchical controller learning in a first-person shooter', in *2009 IEEE Symposium on Computational Intelligence and Games*, p. nil, (9 2009).

[13] Steven Woodcock, 'Game ai: The state of the industry', *Game Developer*, **6**(8), 34–9, (1999). http://www.gamasutra.com/view/feature/131778/game_ai_the_state_of_the_industry.php.