Modelling Human-like Behavior through Reward-based Approach in a First-Person Shooter Game

Abstract. We present two examples of how human-like behavior can be implemented in a model of computer player to improve its characteristics and decision-making patterns in video game. At first, we describe a reinforcement learning model, which helps to choose the best weapon depending on reward values obtained from shooting combat situations. Secondly, we consider an obstacle avoiding path planning adapted to the tactical visibility measure. We describe an implementation of a smoothing path model, which allows the use of penalties (negative rewards) for walking through "bad" tactical positions. We also study algorithms of path finding such as improved I-ARA* search algorithm for dynamic graph by copying human discrete decision-making model of reconsidering goals similar to Page-Rank algorithm. All the approaches demonstrate how human behavior can be modeled in applications with significant perception of intellectual agent actions.

Keywords: Human-like Behavior, Game Artificial Intelligence, Reinforcement Learning, Path Planning, Graph-based Search, Video Game

1 Introduction

The development of video games always face the problem of creating believable non-playable characters (NPC) with game artificial intelligence adapted to human players. The quality of NPC's model in terms of game behavior extremely depends on an interest in the gameplay as in-game interaction of human players with game environment and NPCs. The main entertainment of many games consists of challenging enemy NPCs, so called, BOTs. Human players, on one hand, estimate BOTs to behave like humans, on the other hand, there should be high probability to mine BOT's patterns finding its weaknesses. Human players always estimate the possibility to overcome computer player through intelligence supremacy. The combination of such beliefs is what makes a gameplay interesting and satisfying humans ambitions, but also providing new cognitive field of learning through reward based winning policy.

A first-person shooter game is a special genre of video games simulating combat actions with guns or projectile-based weapons through a first-person perspective. The human player experiences virtual world and action gameplay through the eyes of player's human-like model placed in virtual 3D scene, which is shown at the Figure 1. The problem aroused from the player's expectations of computer players to obtain information from virtual world in a similar way.

Fig. 1. First-Person Shooter Game

From the human point of view it is unfair to have an access to special features and information about game environment, which could not be available and processed by human players during a game. In [1] authors stated the principle that it is better to play against BOTs "on equal terms", rather than against "God-mode" undefeatable opponents. Thus, we aim to make behavior of BOTs similar to human players' behavior in first-person shooter (FPS).

The main criterion of evaluating the quality of a game artificial intelligence is the level of compliance for NPC actions with respect to ability of human experts to distinguish computer-controlled and human players in common and specific in-game situations. One of approaches consists of interpretation of such quality based level of BOT humanness through Alan Turing test for computer game BOTs [2]. In the competition, computer-controlled BOTs and human players that are also judges take part in combat actions during several rounds, whereby the judges try to guess which opponents are human. In a breakthrough result, after five years[1] of attempting from 14 international research collectives, two teams have succeeded in breaking through 25% human-like player behavior barrier. Researchers believed that methods developed for a game A. Turing test should eventually be useful not just in developing intelligent games but also in creating virtual training environments. Both teams separately cracked test with two prototypes of human-like BOTs that try to mimic human actions with some delays and use neuro-evolution model under human gameplay constraints [3]. The disadvantage of such an approach consists of the fact that such models only imitate human intellect but do not give BOT its own cognitive model. In such a case we still do not know what are the reasons for human actions and how BOT could retrieve new information from human gameplay.

However, the most common ways to implement game AI are still finite-state machines and rule-based systems applied to BOTs behavior [4,5]. The cheapest

[1] http://botprize.org/publications.html

way for game developing company is to script behavior of NPCs with respect to restricted game situations fully describing most common NPC actions but not giving it a freedom of choice or enough quantity of randomness in decision making. However, this approach has several serious disadvantages: developers can not script or predict all the possible game situations which may arise during the game, so it is impossible to write all patterns of the rules or the states for NPC behavior. As a result, in a certain game situations BOTs do not act optimal and become recognizable for the wrong decision templates from its scripted model, which significantly reduces the quality of gameplay. This could also lead to BUGs' appearance (semantical and technical errors in BOT's actions).

The idea of selecting script parameters via machine learning are now interesting for the researchers, which could study evolved systems based on rule-based systems [6]. Still, even the BOT model tweaked behavior can not be modified during online game testing without decreasing its quality and stability. The problem also appears when such programmed behavior seems to be static and is not sensitive to changes in the environment and game strategies of other players and their skills' levels.

The authors of [7] present another method for online interactive Reinforced Tactic Learning in Agent-Team Environments called RETALIATE. The system take fixed individual BOT behaviors (but not known in advance) as combat units and learns team tactics rather coordinating the team goals than controlling individual player's reactive behavior. Another real-time behavioral learning video game NERO was presented in [8]. The state-of-art researches on evolution approach can be found in [9,10,11,12].

Following empirical study of machine learning and discrete optimisation algorithms applied to modeling player behavior in a first-person shooter video game[2] we focus on some aspects of human decion-making, such as *weapon selection*, *path planning* and *incremental path finding*. Each section of the paper contains one example of AI improvement based on human behavior, thus creating intensified cycle of applying human behavioral patterns to model them in game.

2 Weapon Selection

Considering the methods of machine learning, such as supervised, unsupervised and reinforcement learning, the latter one gives us the most suitable way to implement BOT's behavior in FPS game. During the process of reinforcement learning BOT receives an award for each committed action, which allows him to accumulate an experience of various game situations and to act in accordance with the collected knowledge, constantly modifying its tactical and strategical decisions [1].

A weapon selection tactics for the BOT should be similar to human player's. In real life we often could not predict the result of an action that we are going to

[2] http://ftp.cs.wisc.edu/machine-learning/shavlik-group/geisler.thesis.pdf

perform. Humans' decisions are based on their personal experience. So, the agent interacts with the environment by performing some actions and then receiving reward from the environment. The purpose of this method is to train the agent to select actions in order to maximize reward value dependently on environment states. In such a model BOTs will choose the most effective weapons with respect to computed parameters of the game environment.

We apply a weapon selection model that is based on neural network from [13]. FALCON (Fusion Architecture for Learning, Cognition, and Navigation) is a self-organizing neural network that performs reinforcement learning. The structure of neural network that we used comprises cognitive field of neurons (it can be also named a category field) and input field (sensory field) that is designed for representing states of environment. It is shown at the Figure 2 Neurons of input fields are connected to neurons of a cognitive field by synapses.

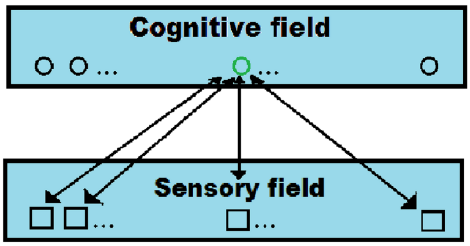

Fig. 2. FALCON Architecture

FALCON enables BOT to remember value of the reward that was received by the BOT when it used some weapon in a particular environment state and use this information to select effective weapons in future.

As of today, we use the values of distance between the BOT and the enemy and percentage of visibility of the enemy as state parameters; the set of weapons that accessible to BOT includes rifle, shotgun, machinegun and knife. Each of the weapons has advantages and disadvantages. We use normalized value of damage that the BOT inflicts to the enemy as reward value.

We modified training algorithm of FALCON in order to make BOT's behavior more human-like. Algorithm of teaching each neuron that we developed consists of two stages. First stage is matching current neuron J from cognitive field and two vectors: vector of probabilities to use each weapon and vector of expected rewards, obtained by the use of a weapon. Every time neuron J is chosen during the process of neuron competition [13], it triggers script of weapon selection

based on random experiment. After the selected weapon is used and reward value is received from environment, algorithm recalculates the expected reward value for this weapon, then recalculates probabilities of the use of weapons and modifies synaptic weights of neuron J. After neuron J was chosen the exact number of times (equal to the neurFixLim parameter value), algorithm considers the neuron qualified enough to be fixed. Next comes the second stage – the period of neuron exploiting. Starting from this moment, every time neuron J is chosen, algorithm selects and then uses the weapon that has the highest probability of use after the first stage. Synaptic weights of the neuron are not modified at this stage. Thus the modified algorithm enables BOT to explore environment and exploit accumulated knowledge in parallel.

Let x denote the input vector; w_j is a vector of synaptic weights of neuron j. Initially, the cognitive field is empty. Iteration of training modified FALCON is described below:

1. For each neuron j of Cognitive field the value of Choice function is computed:
$$T_j = \frac{|\, x \cap w_j|}{w_j}$$
(where the operation $\cap$ is defined by $(\pi \cap \mathrm{q}_i) = \min(\pi, \mathrm{q}_i)$, the norm $|\cdot|$ is defined by $|\mathrm{p}| = \sum i \ \pi$ for vectors p and q).
2. Neurons of Cognitive field are sorted in decreasing order of the value T_j.
3. For each neuron j of Cognitive field condition of vigilance-criterion is checked:
$$\frac{|x \cap w_j|}{|x|} \geq p$$
If the condition is not violated for neuron J, neuron J is selected. If there is no neuron j with not violated condition in the Cognitive field, a new neuron is added to Cognitive field and selected.
4. If selected neuron J is not fixed (that means that it is currently at the first stage of training):
   (a) weapon I is selected via random experiment; then it is used and reward value "r" is received from environment
   (b) weight coefficients of neuron J are modified:
$$w_{j_k} := \frac{n-1}{n} \cdot w_{j_k} + \frac{1}{n} \cdot x_k$$
   (c) Expected reward value of using weapon I for neuron J is modified:
$$expectedReward_I = \frac{expectedReward_I + r}{2}$$
   Probabilities of using weapons for neuron J are modified:
$$p_i^j = \frac{expectedReward_I}{\sum_k expectedReward_k}$$
   Else (if neuron J has been already fixed), a weapon I that matches neuron J is selected and used.

It can be concluded that complexity of each training iteration is $O(n \cdot \log(n))$, in comparison to FALCON's complexity is $O(n^2)$, whereas n is the size of cognitive field.

The results of experiments for one hundred of weapon usages are shown in the Table 1.

Table 1. Original/Modified FALCON

Weapon	Successes, %	Average Reward
Knife	52/73	0.52/0.73
Shotgun	63/79	0.13/0.16
Machinegun	56/68	0.09/0.11
Rifle	54/71	0.08/0.11

The example of a modified FALCON showed us that neural network based on the FALCON can be applied to human-like selecting effective weapons by BOTs during the battle in first-person shooter.

3 Path Planning and Path Finding

Path planning and path finding problems are significant in robotics and automation fields, especially in games. There is a major number of approaches for path planning, such as [14], [15], [16], [17], [18].

The first strategy of path planning is connected with providing believable trajectory of BOT motion to a fixed goal under some constraints. In game programming, Voronoi diagrams (k-nearest neighbour classification rule with $k = 1$) are used to make a partition of a navigation mesh to find a collision free path in game environments [14,17,19]. Smooth paths for improving realism of BOT motion are made through splines [20,21] or by using Bezier curves [15,17,22,23]. We used combined approach of both smoothing methods following the works of [15,24,25].

The second strategy of path planning consists of computing tactical properties of a map as a characteristic of Voronoi regions areas. We compute offline tactical visibility characteristics of a map for further path finding penalties and frag map usage to transform paths found by the first strategy to optimise certain game criteria.

The navigation starts with BOT's query to navigation system. Navigation system uses path finding algorithm I–ARA* anytime algorithm from [26] to obtain a sequence of adjacent polygons on navigation mesh. Then a sequence of polygons is converted into a sequence of points. Finally, BOT receives a sequence of points and build a collision free path to walk. We design the interface for an interaction between a querier and the navigation system at each iteration of A^* algorithm. We use region parameters to manage penalties for path's

curvature, crouching and jumping at the current Voronoi cell. There is also a special method for querier to influence the navigation with respect to previous movement direction, similar to Markov's chains in path planning published by authors . We also used general penalties, such as base cost, base enter cost and no way flag, which can be dynamically modified by any game event.

Now we describe a family of path finding algorithms and how we could use modelling human behavior to reduce their complexity. In contrast to the Dijkstra's algorithm, A* target search uses information about the location of a current goal and choose the possible paths to the goal with the smallest cost (least distance obtained), considering the path leading most quickly to the goal.

Weighted A* as it was presented in [27] was a modified algorithm of A* search with the use of artificially increased heuristics, which leads to the fact that the found path was not optimal. The improvement of these algorithms is ARA* [28]. The purpose of this algorithm is to find the minimum suboptimal path between two points in the graph under time constraints. It is based on iterative running of weighted A* with decreasing to 1 heuristics values. If it decreases exactly to 1, then the found path is optimal.

Algorithm I-ARA* works as well as repeated ARA*, with the only difference that it uses the information from the previous iteration [29]. The first search made using I-ARA* is simple ARA* search.

We present a modification of I-ARA* as human discrete optimisation decision-making: rather than looking at each step for a new path to the target we simply walk proposed suboptimal path until we passed a certain part (partial path length) from the previously found path. The larger the distance, the longer the last iteration I-ARA*, so most of the time-consuming iterations of this algorithm could be omitted. As a result, we found that the number of moves in modified and original I-ARA* algorithms differs not greater than 10% in average but time for computation has been significantly reduced by 5-20 times when labyrinth has not extremely dense wall structure.

For proper work of I-ARA* algorithm, each penalty is jammed to a limited range, so the resulting penalty is not less than the Euclidean distance, which is used as heuristics in our implementation. Once a path is found, it should be converted into a point sequence.

We generated 2D mazes with sizes of 300 by 300 and 600 to 600 with density of free cells equaled to 0.1, 0.2, 0.3, 0.4. For every field size 100 trials have been conducted. During each test, we choose 30 pairs of random free cells and test the value of the heuristic P as percentage of a path length to go until next computation will be needed. In the Table 2 we presented the results of searching path time decreasing (%) and path length increasing (%) for modified I-ARA*. It is easy to see that for dense mazes our modification significantly wins in time with path length stabilizing or even shortening. For sparse mazes increasing of P leads to the error increasing.

When developing a BOT navigation, *smoothing* is one of the key steps. It is the first thing for a human to distinguish a BOT from a human player. Several approaches can be used to smooth movements. *Bezier curves* seem to be the

Table 2. Time decreasing/Path Length Increasing Comparison

Sparseness	P=0.05	P=0.1	P=0.15	P=0.2	P=0.25	P=0.3	P=0.35
0.1	785/-.06	1194/-.40	1483/0.07	1744/-.64	1965/0.33	2238/0.83	2354/0.87
0.2	293/0.20	410/0.72	526/1.54	578/2.55	666/2.55	725/2.37	785/4.64
0.3	283/2.20	398/2.25	476/2.11	540/4.24	610/6.53	624/7.53	664/10.71
0.4	221/0.06	309/0.39	346/0.62	379/1.75	406/1.52	395/7.72	419/11.94

most suitable because they could be represented as a sequence of force pushes from obstacles guaranteeing that BOT will not be stuck into an obstacle.

In practice, the contribution of visibility component to remain undetected during BOT motion is very low if we are not taking into account the enemies' movements. We consider the relative dependence of the smooth low-visibility path length with the length of the shortest path obtained by Recast navigation mesh. The resulting difference between the smooth paths with and without a visibility component does not exceed 10–12% , that was shown by authors in the other research article, so taking into account tactical information seems to be a useful decision. The difference in 15–25% between smooth path length from our algorithm and the results from [24,25] is not too significant because we mainly focus on constructing realistic randomized paths for BOTs. We also create OWL reasoner to choose whether we have to use smoothing or piece-wise linear path to answer query for current combat situation like it is shown at the Figure 3. When implementing such an algorithm in 3D first-person shooter, we obtained

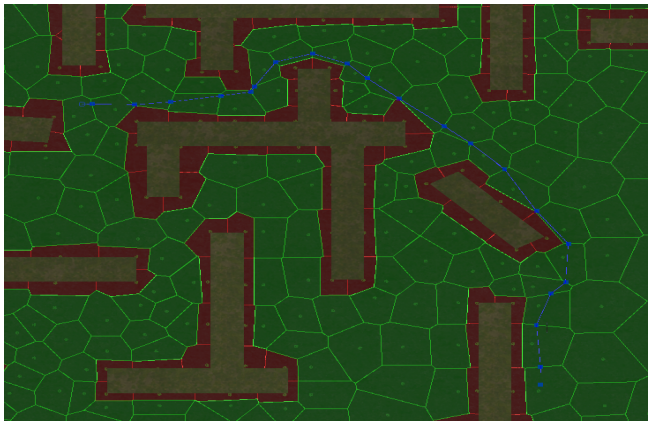

Fig. 3. Path finding

more realistic motion behaviours than the minimized CBR-based path, while saving the property of the path to be suboptimal.

4 Conclusion

We started our research with stating the thesis that modeling human behavior in video games could be presented as game artificial intelligence problem that should be implemented by algorithms with human patterns of discrete optimisation. We used obvious assumptions on neuron to be useful in terms of short memory usage to balance neural network. Smoothing path trajectory was obtained through a native obstacle avoidance model supporting enough degree of randomness. Path finding algorithm with reduced time computations was obtained from discrete choice model used by human players (firstly implemented as the first and the simplest game AI for ghost-BOT in computer game PACKMAN). We hope that idea to use the simplest optimisation criteria from the Occam's razor to model human behavior in video games is a key to understanding correct reasoning of models containing information about evolution of decision-making models while increasing its game experience.

References

1. Wang, D., Tan, A.H.: Creating autonomous adaptive agents in a real-time first-person shooter computer game. IEEE Transactions on Computational Intelligence and AI in Games **7**(2) (June 2015) 123–138
2. Hingston, P.: A turing test for computer game bots. IEEE Transactions on Computational Intelligence and AI in Games **1**(3) (Sept 2009) 169–186
3. Karpov, I.V., Schrum, J., Miikkulainen, R. In: Believable Bot Navigation via Playback of Human Traces. Springer Berlin Heidelberg, Berlin, Heidelberg (2012) 151–170
4. van Hoorn, N., Togelius, J., Schmidhuber, J.: Hierarchical controller learning in a first-person shooter. In: 2009 IEEE Symposium on Computational Intelligence and Games. (Sept 2009) 294–301
5. da Silva, F.S.C., Vasconcelos, W.W. In: Rule Schemata for Game Artificial Intelligence. Springer Berlin Heidelberg, Berlin, Heidelberg (2006) 451–461
6. Cole, N., Louis, S.J., Miles, C.: Using a genetic algorithm to tune first-person shooter bots. In: Evolutionary Computation, 2004. CEC2004. Congress on. Volume 1. (June 2004) 139–145 Vol.1
7. Smith, M., Lee-Urban, S., Muñoz-Avila, H.: RETALIATE: learning winning policies in first-person shooter games. In: Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22-26, 2007, Vancouver, British Columbia, Canada, AAAI Press (2007) 1801–1806
8. Stanley, K.O., Bryant, B.D., Miikkulainen, R.: Real-time neuroevolution in the nero video game. IEEE Transactions on Evolutionary Computation **9**(6) (Dec 2005) 653–668
9. Veldhuis, M.O.: Artificial intelligence techniques used in first-person shooter and real-time strategy games. human media interaction seminar 2010/2011: Designing entertainment interaction (2011)
10. McPartland, M., Gallagher, M.: Reinforcement learning in first person shooter games. IEEE Transactions on Computational Intelligence and AI in Games **3**(1) (March 2011) 43–56

11. McPartland, M., Gallagher, M.: Interactively training first person shooter bots. In: 2012 IEEE Conference on Computational Intelligence and Games (CIG). (Sept 2012) 132–138
12. McPartland, M., Gallagher, M. In: Game Designers Training First Person Shooter Bots. Springer Berlin Heidelberg, Berlin, Heidelberg (2012) 397–408
13. Tan, A.H.: Falcon: a fusion architecture for learning, cognition, and navigation. In: Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on. Volume 4. (July 2004) 3297–3302 vol.4
14. Bhattacharya, P., Gavrilova, Marina L.: Voronoi diagram in optimal path planning. In: 4th IEEE International Symposium on Voronoi Diagrams in Science and Engineering. (2007) 38–47
15. Choi, J.w., Curry, Renwick E., Elkaim, Gabriel H.: Obstacle avoiding real-time trajectory generation and control of omnidirectional vehicles. In: American Control Conference. (2009)
16. Gulati, S., Kuipers, B.: High performance control for graceful motion of an intelligent wheelchair. In: IEEE International Conference on Robotics and Automation. (2008) 3932–3938
17. Guechi, E.H., Lauber, J., Dambrine, M.: On-line moving-obstacle avoidance using piecewise bezier curves with unknown obstacle trajectory. In: 16th Mediterranean Conference on Control and Automation. (2008) 505–510
18. Nagatani, K., Iwai, Y., Tanaka, Y.: Sensor based navigation for car-like mobile robots using generalized voronoi graph. In: IEEE International Conference on Intelligent Robots and Systems. (2001) 1017–1022
19. Mohammadi, S., Hazar, N.: A voronoi-based reactive approach for mobile robot navigation. Advances in Computer Science and Engineering **6** (2009) 901–904
20. Eren, H., Fung, C.C., Evans, J.: Implementation of the spline method for mobile robot path control. In: 16th IEEE Instrumentation and Measurement Technology Conference. Volume 2. (1999) 739–744
21. Magid, E., Keren, D., Rivlin, E., Yavneh, I.: Spline-based robot navigation. In: International Conference on Intelilgent Robots and Systems. (2006) 2296–2301
22. Hwang, J.H., Arkin, R.C., Kwon, D.S.: Mobile robots at your fingertip: Bezier curve on-line trajectory generation for supervisory control. In: IEEE International Conference on Intelligent Robots and Systems. Volume 2. (2003) 1444–1449
23. Škrjanc, I., Klančar, G.: Cooperative collision avoidance between multiple robots based on bezier curves. In: 29th International Conference on Information Technology Interfaces. (2007) 451–456
24. Ho, Y.J., Liu, J.S.: Smoothing voronoi-based obstacle-avoiding path by length-minimizing composite bezier curve. In: International Conference on Service and Interactive Robotics. (2009)
25. Ho, Y.J., Liu, J. S.: Collision-free curvature-bounded smooth path planning using composite bezier curve based on voronoi diagram. In: IEEE International Symposium on Computational Intelligence in Robotics and Automation. (2009) 463–468
26. Koenig, S., Sun, X., Uras, T., Yeoh, W.: Incremental ARA*: An incremental anytime search algorithm for moving-target search. In: Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling. (2012)
27. Pohl, I.: First results on the effect of error in heuristic search. Machine Learning **5** (1970) 219–236
28. Likhachev, M., Gordon, G., Thrun, S.: ARA*: Anytime A* search with provable bounds on sub-optimality. In Thrun, S., Saul, L., Schölkopf, B., eds.: Proceedings of Conference on Neural Information Processing Systems (NIPS), MIT Press (2003)

29. Sun, X., Yeoh, W., Uras, T., Koenig, S.: Incremental ara*: An incremental anytime search algorithm for moving-target search. In: ICAPS. (2012)