Implementing AI for Non-player Characters in 3D Video Games

Chapter *in* Lecture Notes in Computer Science · February 2018

DOI: 10.1007/978-3-319-75417-8_57

Marek Kopel
Wrocław University of Science and Technology

Implementing AI for non-player characters in 3D Video Games

Marek Kopel and Tomasz Hajas

Faculty of Computer Science and Management,
Wroclaw University of Science and Technology,
Wybrzeze Wyspiaskiego 27, 50-370 Wroclaw, Poland
`marek.kopel@pwr.edu.pl,tomaszhajas@gmail.com`
`http://ksi.pwr.edu.pl/kopel`

Abstract. The purpose of this work was to find a solution for implementing intelligent behavior of independent NPC agents (non-player characters) in video games. NPC is a computer operated character - usually an enemy to the human user player. In modern video games NPCs are programmed to mimic human player behaviour to increase realism. Four approaches to NPC AI implementation were compared: decision tree, genetic algorithm, Q-learning, and a hybrid method. Results were aggregated and discussed along with recommending the best approach.

Keywords: artificial intelligence, video game, non-player character, neural network, genetic algorithm, Q-learning, Unreal Engine

1 Introduction

The key element of realism in video games is the behavior of computer controlled characters. Gameplay becomes unique, when artificial intelligence (AI) is a part of the immersive, virtual environment. In video games, every snippet of code that targets simulating ”intelligent” behavior of virtual players is considered AI. This behavior on its own doesn’t have to be sophisticated - its only real role is to fulfill potential player’s expectations by providing adequate level of entertainment. Such software is most commonly based on a mere illusion of intelligence produced by a skillful usage of game design techniques, which harness simple controlling algorithms, realistic graphics, convincing character animations and voices borrowed from famous film actors. Often video game characters are controlled by a global algorithm, which has access to all variables inside the program and unjustly uses this knowledge to beat user player.

On average, AI created by video game developers is less sophisticated than techniques used in academic and industrial environments. [1] claims it happens mainly because of the following reasons:

– lack of CPU resources available to AI,
– suspicion in the game development community of using non-deterministic methods,

- lack of development time,
- lack of understanding of advanced AI techniques in the game industry,
- fact that efforts to improve the graphics in games usually overshadows all else, including AI.

Today, more and more popularity is gained by goal-driven artificial intelligence. It is a system for computer agents (characters), which allows them to plan an action sequence needed for achieving a given goal. Such sequence not only depends on the goal, but also on agent's actual state and environment. This means, that when two agents with different states share the same goal, they can generate two totally different action sequences. Such approach makes AI more dynamic and realistic.

1.1 Non-player character

In video games, non-player character (NPC) is a computer operated character - usually an enemy to the human user player. In modern first person shooters (FPS) and other action games they are programmed to mimic human players [10] to increase realism. But their tradition go a long way back to first arcade games, like Space Invaders, where opponents moving pattern is getting more complex with each level. In massively multiplayer online role-playing games (MMORPGs) these characters are usually called mobs (from mobile objects) or bots. This is where dumb mobs were commonly used. They had no complex behaviors beyond attacking or moving around. Their implementation was a *static* model with a pre-made waypoints for each map that they followed. More recent implementations in both RPG and FPS also combine *dynamic* approach, which allows for hunting state. In this state NPC is actively looking for enemy traces with pathfinding and learning the map in real time. Complex behaviour of NPC based on decision making policy is usually implemented using heuristic, artificial intelligence methods and called AI. Early bot AI method - usually because of performance reasons - was cheating. This means bots were allowed actions and access to information that would be unavailable to the player in the same situation (e.g. the user player position). This technique is considered acceptable as long as the effect is not obvious to the player. But using heuristic AI will always give more human-like behaviour. As Simon Herbert states in [9], AI always had its niche in area, where computer programs use heuristic searches (which he names "the way of human thinking"), without guarantee of complete results, often using only sufficient success criteria.

1.2 Related works

According to [8], state-of-the-art AI is done mostly using deep learning (DL) and neural networks (NN). But its application to games is still at the 8-bit console stage. Modern 3D games and character behaviours are yet too complex for current AI performance. Google Deepmind implementation of deep reinforcement learning (RL) is showcased in [5] by AI that is learning to play 8-bit Atari 2600 video games. Author of [4] created Arcade Learning Environment (ALE) allowing testing different approaches for achieving human-level performance in Atari 2600 emulator. The same way NN and genetic algorithm (GA) were successfully used in 16-bit platformer Super Mario World. MarI/O is SNES emulator AI plugin implemented by YouTuber SethBling using NEAT method described in [11]. It can outperform human in beating the game.

Earlier works on using RL for both 2D strategy and 3D FPS games, could not create NPC that would outperform human. However the goal there is different. According to [10] in FPS, like Unreal Tournament the goal is to make NPC act similar to human characters. In [2] the AI was applied to NPC in tank battle game. In [13] RL is used to create NPC Team playing Unreal Tournament. As claimed by authors: the ultimate goal of game AI is to enhance entertainment, not to develop invincible NPCs, because a game you cannot win is no fun. In [6] authors create Genetic Bots, using GA and genetic programming, to outperform default bot AI in Unreal.

One way to deal with AI in modern games is to use a common framework - AI middleware. According to [7] this idea implementations can be found in game engines like Unity, Ureal Engine, CryEngine, and Havok. In popular game mashup platform Garrys Mod ([12]) the default way to create NPC is NextBot - the common model used as AI in games Team Fortress 2 and Left 4 Dead.

2 Method

The method tested as NPC AI starts as a deterministic decision tree (DT) based on finite state machine (FSM) concept. NN is first learned to mimic DT behaviour and then evolves using GA to find better solutions, achievable by DT. GA is a common algorithm to use for control decision making. But using it by itself shows it can stuck at local minima. This is where enhancing it with Q-learning (QL) becomes helpful. Decision making system is modeled as multilayer perceptron (MLP) NN with one layer of hidden neurons. The network uses feed-forward technique to process inputs from agent (NPC) sensors. The sensors feed the network with information like: is NPC in the air/on the ground, can NPC see an enemy, which way and how far is the enemy, what was last action of NPC, etc. NN output is next NPC action selected from predefined set: (turn left/right, move forward, atack, jump).

2.1 Experiment setup

The model and all tested algorithms are implemented in Unreal Engine (see figure 1) using blueprints - a way for programming logic with graph diagrams. An example blueprint is presented in fig. 2. The algorithms used (DT, GA, QL) are implemented in their basic, most popular versions. For the reinforcement the QL function is iterated as shown in formula 1. It is defined as the reward observed for the action A in current state S plus a fraction $\gamma \in < 0, 1 >$ of reward for the next state S' achieved by an optimal action A'.

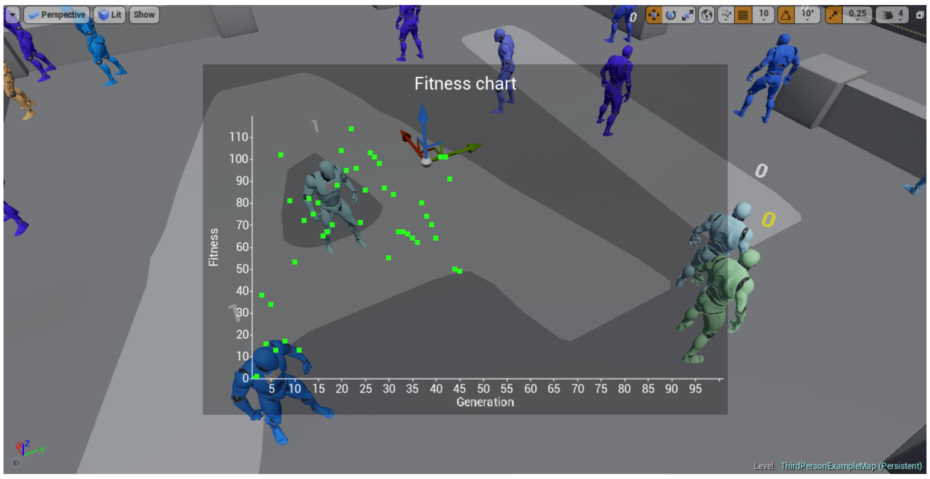

Fig. 1. Graphical interface of the research prototype, showing real time chart with fitness function for all NNs (for each NPC) through progressing generations. Behind the diagram a real time simulation is taking place with NPCs moving and interacting on a level map. Over each NPC head his current fitness function score is shown.

$$Q(S, A) = r + \gamma \cdot \max(Q(S', A')) \tag{1}$$

During the simulation the engine was running on 64-bit system using Intel Core i7-4810MQ CPU @ 2.80 GHz with 8 GB of RAM. This configuration allowed accelerating the simulation time and make more test possible.

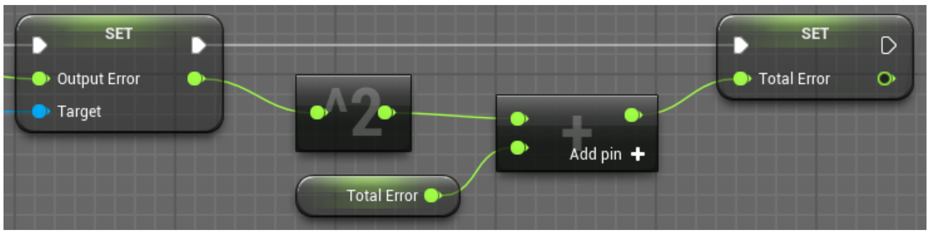

Fig. 2. An example part of blueprints. In Unreal Engine blueprints serve as visual way for representing and developing in-game logic. The algorithms and the NN model researched in this paper were implemented entirely using blueprints.

3 Results

During the experiment four approaches to NPC AI have been implemented. The approaches concern using DT, GA, QL, and a hybrid of GA+QL. Two main measures are used to compare the approaches:

1. learning speed of NPC NN in each generation (presented in figure 6)
2. fitness function values for each generation of NPCs (presented in figures 3 and 4),
3. total squared error of each NPC in each generation (presented in figure 5).

Fitness function values in figures 3 and 4 are plotted in 3 series with distinctive colors:

– Best - series for best individual in each generation,
– Average - series for average score of all individuals, in each generation
– Worst - series for worst individual in each generation.

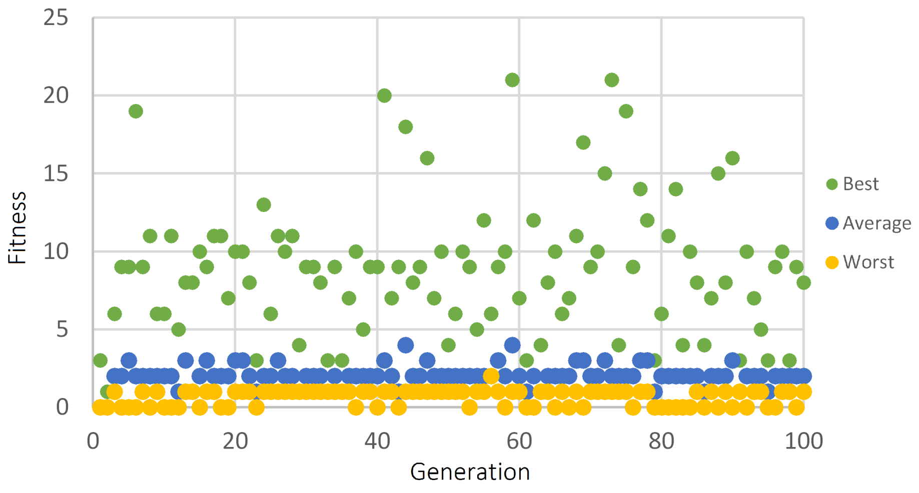

Fig. 3. Fitness function values of individual NPCs using GA approach.

Individuals trained with QL method because of processing only their own states do not have a possibility of predicting other character movements. However, upon considering built-in limitations of recursion tree depth, this flaw does not seem substantial.

NN was able to imitate function Q even after 20 generations of individuals (which lasts 400 seconds in unaccelerated simulation). It was susceptible to overfitting (or overlearning) phenomenon, resulting in increasing the squared error over time. As stated in [3], overfitting is caused by too accurate adjustment of model to learning data. In this case the learning data are created in real time with QL technique. The more overfitted model is, the more damage causes a single neuron weight mutation. The main bottleneck here are values of weights oscillating near 0 - even a minor change (up or down) can end with rapid changes in general classification-related traits of the NN. Overlearned (overfitted) individuals are characterized by lower adaptability and susceptibility to minor model changes, e. g. genetic manipulations or fitness function change.

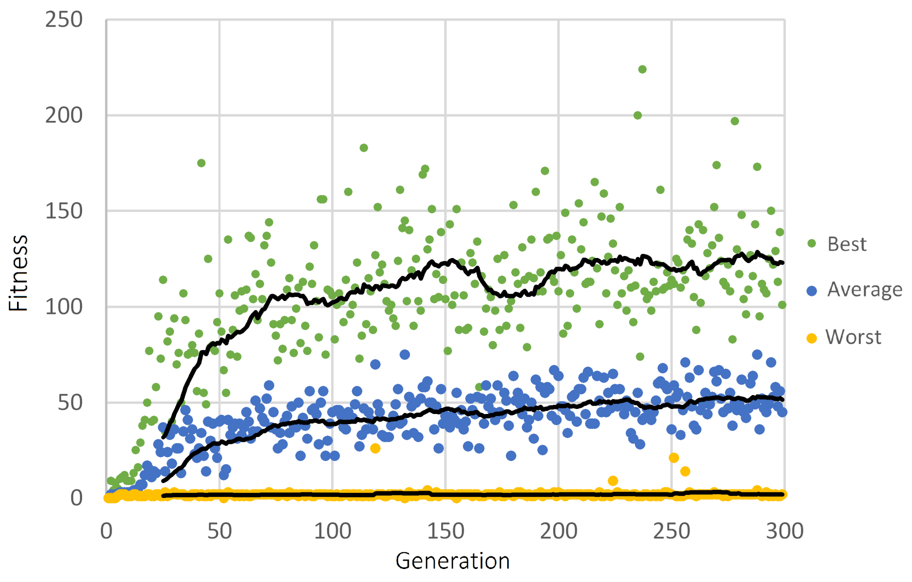

Fig. 4. Fitness function values of individual NPCs using GA+QL hybrid method. QL was stopped in generation 165. Independent GA with success is able to elevate average fitness value, and sometimes reaches even higher levels of evaluation of best individual. Near generation 230 AI instances periodically became similar to each other (GA trait), making fitness values equal. The moment of changing learning policy was empirically unnoticeable. Black lines mark trend lines for corresponding data series (average of 25 neighboring points)

Early stopping method was used in order to avoid model overfitting. This method was creating backups of NNs just after observing significant increase of total squared error (see figure 5). Overfitting not always caused noticeable quality drop in agents behavior - sometimes GA was creating solutions, which were recognized by strict QL as much worse, although to a human observer (and judging by fitness function values) they were still performing very well.

In order to prevent limitations of evolutionary potential, a decision was made to turn off the weight back-propagation implemented in QL algorithm, just after achieving low and stable total squared error values. GA left alone has partially substituted QL, increasing fitness of individuals even further (by making them diverse and selecting the best networks).

Fitness function charts for networks learning with QL technique were characterized by general increase oscillating around logarithmic curve (see figure 4).

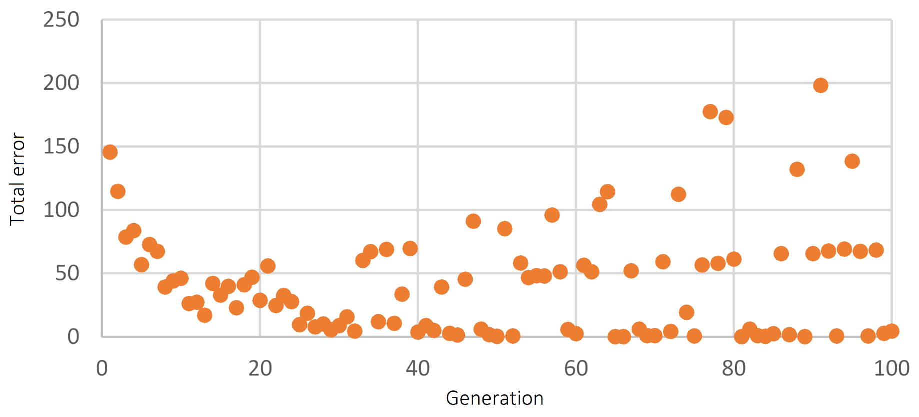

Fig. 5. Chart presents total squared error of every network in each generation in evaluation of hybrid method (GA with QL). Since generation 32, when model already learned to mimic decision tree, oscillations between two rising levels are observable, which are the result of network overlearning. Each rising level is caused by essential neurons weight random mutation (small value change, up or down). Third, lowest level is composed of generations without essential weight mutation inside one of the networks.

4 Discussion

Both methods (GA and QL) were compared with each other and then with a hybrid of the two component methods. As a reference, DT method was used, which is also a key component of the QL method.

The first and the most obvious comparing factor was learning speed of NN models. Then maximal values of fitness function were researched as secondary measures.

Each starting population had weights picked at random from the same seed, and individuals were set to make decisions in a deterministic manner (winning output neuron implies final action). It allowed to avoid excessive randomness and construct comparison conclusions. Fitness values for best individuals are shown in figure 7.

Speed of building AI model (learning) with QL was noticeably greater because of a constant flow of data which came from environment (see figure 6). Isolated GA approach needs several times more generations to achieve average

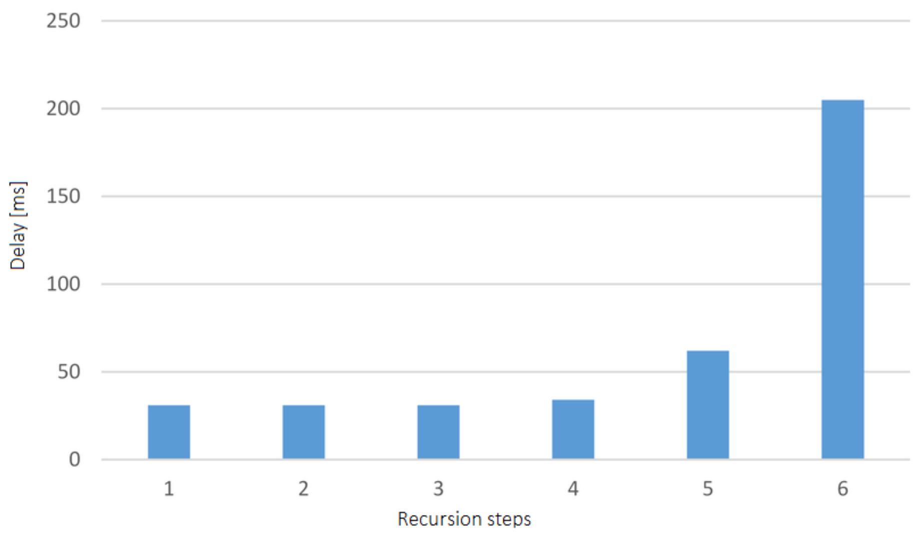

Fig. 6. Average delay (in milliseconds/frame) dependent on recursion depth in Q function. Calculated in the moment of simulations 10 first seconds with player camera directed towards the center of simulation area. PC specs: Processor: Intel Core i7-4810MQ CPU @ 2.80 GHz, RAM: 8 GB, system architecture: 64-bit

fitness values comparable with QL results. The results show that simulation delay caused by learning model is acceptable for GA, QL and their hybrid solution.

Maximal fitness values of individuals in both methods were similar, but in GA approach the achieved values should be associated with a great randomness factor what in figure 3 looks like a bunch of oscillating points.

Using GA caused no performance problems. However, for QL - when used with high number of recursive steps - this was not the case. Number of recursive steps in quality function had a great impact on the performance of learning process. Maximum reference value for simulation execution delay was 50 milliseconds popular value accepted among PC gamers (also in network games). As shown in figure 6, the maximum number of steps, which allows to run calculations in real time, equals 4. Further increasing of recursion depth was resulting in severe delay, which interferes with simulation clock and disables potential gameplay.

5 Conclusions

In this paper a comparison of popular methods for intelligent behavior of non-playable video games characters is presented. Findings show that, best results can be achieved using a hybrid method consisting of multilayer perceptron NN with QL rule and GA.

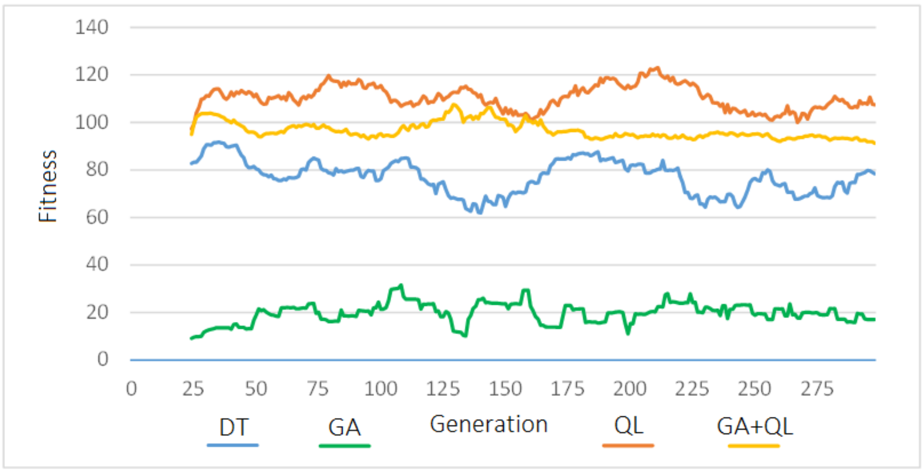

Fig. 7. Fitness function values for best individuals (trend lines of moving average including 25 closest points). Learning algorithms were compared with simple decision tree, which during the learning took a role of Q function. QL algorithm and hybrid solution with GA outperformed decision tree with fitness function values. Sole GA in opposition to QL did not expose any ability to develop complex NPC behavior.

Set of methods proposed in this paper has potential to recognize environmental patterns and execute near-intelligent actions. It demonstrates also self-organizing function, which is a feature of unsupervised learning. NN left alone after a while learns useful behavior patterns, which can be seen as making progress in a simulation. A goal achieved this way is analogical to the one achieved by a simple, decision tree-based control algorithm. But this one is enriched with adaptability aspect, common for NNs.

During the experiments, QL properties has shown learning acceleration feature, but its potential is limited to a defined Q function. It makes sense to use QL with GA only at the beginning of a simulation, and then - after encountering fitness function stagnation - switch process to sole GA.

Method efficiency was demonstrated in computer simulations, which lifespan (in the most of cases) was not longer than one hour. Because of relatively low delay, algorithms used in learning virtual agents (NPCs), can be used in real-time video games.

Despite multiple operation types and computational complexity, the method is still far from imitating the behavior of living creature neurons. Nevertheless, it makes a good base for creating relatively simple AI featuring learning traits.

Big number of various input sensors of agent NN implied great deterioration of AI model learning speed. Learning process with a small number of binary sensors was progressing much quicker.

Further steps on the research shall be taken. Effectiveness of learning algorithms with no doubt could rise after switching the model architecture into 3rd

generation NN spiked NN with impulse-driven neurons. It should be also beneficial to use recursive NN, dedicated to multi-state, sequence solutions. Such sequences can be surely found in 3D computer environment. However it should be reminded that the optimal solution is a consensus between accurate representation of human behavior and device-dependent computational performance.

References

1. Fairclough, C., Fagan, M., Mac Namee, B., Cunningham, P.: Research directions for ai in computer games. Tech. rep., Trinity College Dublin, Department of Computer Science (2001)
2. Fang, Y.P., Ting, I.H.: Applying reinforcement learning for game ai in a tank-battle game. In: Innovative Computing, Information and Control (ICICIC), 2009 Fourth International Conference on. pp. 1031–1034. IEEE (2009)
3. Gurney, K.: An introduction to neural networks. CRC press (1997)
4. Hosu, I.A., Rebedea, T.: Playing atari games with deep reinforcement learning and human checkpoint replay. arXiv preprint arXiv:1607.05077 (2016)
5. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al.: Human-level control through deep reinforcement learning. Nature 518(7540), 529–533 (2015)
6. Mora-García, A.M., Merelo-Guervós, J.J.: Evolving bots ai in unreal. In: Algorithmic and Architectural Gaming Design: Implementation and Development, pp. 134–157. IGI Global (2012)
7. Safadi, F., Fonteneau, R., Ernst, D.: Artificial intelligence in video games: Towards a unified framework. International Journal of Computer Games Technology 2015, 5 (2015)
8. Schmidhuber, J.: Deep learning in neural networks: An overview. Neural networks 61, 85–117 (2015)
9. Simon, H.A.: Artificial intelligence: an empirical science. Artificial Intelligence 77(1), 95–127 (1995)
10. Soni, B., Hingston, P.: Bots trained to play like a human are more fun. In: Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on. pp. 363–369. IEEE (2008)
11. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evolutionary computation 10(2), 99–127 (2002)
12. Studios, F.: Garrys mod (2006)
13. Wang, H., Gao, Y., Chen, X.: Rl-dot: A reinforcement learning npc team for playing domination games. IEEE Transactions on Computational intelligence and AI in Games 2(1), 17–26 (2010)