Evolving a Character in a First-person Shooter

Chishyan Liaw, Ching-Tsorng Tsai, Chung-Chi Lin and Jing-Long Wu
Department of Computer Science, Tunghai University, Taichung 40799, Taiwan

Keywords: First-person Shooter.

Abstract: This paper presents an effective strategy of evolving a character in the Quake III Arena, a first-person shooter. A genetic algorithm (GA) is used to evolve a character's capabilities and behaviors. In order to increase a character's ability, the GA is used to determine the constrained weights for the behavior control. In the experiments, the GA is implemented to design a non-player character (NPC) which is shown to be superior to the other characters originally created in the Quake III game. This evolving strategy decreases the amount of effort required by game designers to design an intelligent character's behaviors.

1 INTRODUCTION

Choosing a proper set of parameters in a first-person shooter is not a straightforward process. Various research efforts have been devoted to exploring approaches using fuzzy automatons in designing character behaviors (Alexander, 2002); (Zarozinski, 2002). In a fuzzy-state machine, an output result is chosen randomly or according to a complex weight function. In addition to the approach of fuzzified automatons-based approach, other methods are also used. For example, Ponsen et al., (2007) used offline learning to discover effective tactics that can be utilized to beat static opponent strategies. Smith et al. (2007) applied an online reinforcement learning algorithm to develop winning policies in team first-person shooter games with fixed individual agent behavior.

Zanetti and El Rhalibi (2004) introduced machine learning techniques for FPS in the Quake III Arena. The system is able to learn certain behaviors, but still lacks in others. Most of the studies investigated how an NPC can learn from experience of expert human players. The strategies take significant time to evolve. It is very difficult for game developers to decide when and where to deploy certain tactics (Zanetti and El Rhalibi, 2004).

The Particle Swarm Optimization (PSO) was applied to evolve game agents. (Messerschmidt and Engelbrecht, 2004); (Tsai et al., 2011). PSO has the advantage of being able to converge fast; however, a local optimum is likely acquired instead of obtaining a global optimum (Shyr, 2008).

In 1975, Holland introduced the genetic algorithm (GA), which is an evolutionary algorithm that imitates natural evolutionary processes such as inheritance, selection, crossover, mutation, etc., to search for solutions to optimization problems. Genetic evolutionary computations, including genetic algorithms, genetic programming, etc., have been applied to solve nonlinear problems such as optimization, automatic programming, machine learning, economics, etc. (Sipper et al., 2007).

In this paper, GA is used to evolve a character's behaviors in a first-person shooter. The method was implemented on the platform of the Quake III. In the Free for All event of the game, a character is evolved to compete against original opponents. The experimental results show that the evolved character is superior to all the original characters.

2 QUAKE III ARENA - FIRST-PERSON SHOOTER

In the Quake III Arena, the player and NPCs move around in a 3D virtual arena to kill opponents with score points calculated based on the objective of the game event. The game events include Free for All, Tournament, Team Death Match, Capture the Flag, etc.

In the event of Free for All, a player can challenge a number of gladiators who reside in the arena. Each of the gladiators has its own tactics and personality. In the game, everyone is the fragged

Liaw C., Tsai C., Lin C. and Wu J..
Evolving a Character in a First-person Shooter.
DOI: 10.5220/0004112702650267
In *Proceedings of the 4th International Joint Conference on Computational Intelligence* (ECTA-2012), pages 265-267
ISBN: 978-989-8565-33-4

target of others to get score points.

The health of a fighter is displayed as a number. Health is lost if a combatant is wounded or fragged and can be regained by running through the health bonus in an arena. The game ends when the time limit has been reached, or if a combatant reaches a specified score.

A character may have its own strategies, such as attack or retreat in the game. Some tactics may be self-preservative or aggressive. Each tactic consists of a few behaviors. Finding suitable parameters for an intelligent character's strategies and behaviors is of major interest in this paper. These parameters include attitudes used in fighting, offense or defense, attack or ambush, vengefulness or frag easy target, etc. The released Quake III source codes are used for modification. The favorite goals, tactics, and behaviors of a character can be altered in external files. These files will be loaded prior to a game so that the character's actions in the game can be changed accordingly.

3 THE EVOLVING METHOD

This section introduces a strategy of evolving an intelligent gaming character which is able to win in the first-person shooter, Free for All in the Quake III Arena.

3.1 Designing the Behaviors

An AI mechanism is available for players to create a character. The Dynamic Linking Libraries (DLLs) consist of bot AI open source and bot AI library, which can be called by the main program of the Quake III during a game. The external bot files, which specify a character's name, weights setting, and so on, will be loaded by programs of bot AI library before a character joins the game.

The bot AI library is responsible for loading the external bot files, perceiving and acting of a bot, and controlling of certain bot's behaviors. The bot AI open source controls most of a bot's behaviors. In addition, a strategy and an action will be also determined according to the description of a bot's behaviors and parameters in the external bot files.

3.2 Evolution of a Character

A bot's abilities, behaviors, etc. are described in parameters in the external bot files. A genetic algorithm is applied to modify those parameters described in the external bot file based on the progress of a game.

Initial values are randomly generated for those parameters before optimization begins. The character is then allowed to play in a game; at the end of the game, the performance of the character is evaluated and a new set of parameters is decided accordingly. The optimization process is repeated until a stopping rule is met. The behavior parameters are adjusted by the GA approach dynamically to yield the best solution.

3.3 Evolution in the Game

Genetic algorithm is applied to search for an optimal solution. The best agent evolves from inheritance, selection, crossover, and mutation after generations. Initially, randomly-selected individuals, which are possible solutions to the best strategy in a game, are created. Their performance is evaluated based on a preset fitness function after a generation.

The goal in the game is to kill as many bots as possible while trying not to be fragged. The fitness function for bot i is defined as:

$$f_i = \alpha \times k - \beta \times d \quad (1)$$

where k is the number of bots killed by bot i, d is the number of times bot i was fragged to death, and α and β are the weights of kill and death, respectively. Agents with high fitness values can breed their offspring. The evolution process is repeated until an optimal solution is found or a maximum number of generations is reached.

The strategies of an agent's behavior are encoded in a character string. Thus, the chromosomes, which are the behavior in a game, for bot *i* in generation *j* are $B_{i,j}$. The proposed method first generates *n* agents in which chromosomes are given randomly. Each one then takes turns playing against a built-in bot, which is controlled by the Quake III AI. The performance of all agents is then evaluated according to Eq. (1). The chromosomes of *b* best-performing agents are stored in a gene pool. The chromosomes of v new bots are generated by selecting these chromosomes in the pool randomly, and proceeding to a crossover operation. A mutation is completed by changing *t* of the chromosomes generated from the crossover in each bot. The v worst-performing bots in the generation are replaced by the new offspring agents. Thus, the bots of the next generation are bred by inheriting the best-performing agents' chromosomes, as well as their crossover and mutation.

4 EXPERIMENTAL RESULTS

The proposed approach was examined in the mode of the Free for All. The presented evolution system was coded in Visual Basic. The weights for $\alpha=5$ and $\beta=3$.

In the experiment, the character created based on the proposed approach was named Harley. Harley and 5 other built-in bots fought against each other in the games.

In the experiments, each match ends when a bot has killed 15 other bots. The probability of crossover and mutation are 0.4 and 0.1, respectively. We let Harley compete against 5 other original bots for a 30-match game. The results are illustrated in Table 1 and Figure 1. They demonstrate that the evolved agent has better performance than the original characters.

Table 1: Number of times a bot is fragged and number of bots killed.

Bot	Number of death	Death rate %	Number of kills	Kill rate %
Harley	242	12.3	365	22.5
Anarki	302	15.3	262	16.2
Angel	344	17.5	247	15.2
Biker	379	19.2	277	17.1
Bitter	320	16.2	223	13.8
Bones	383	19.4	246	15.2

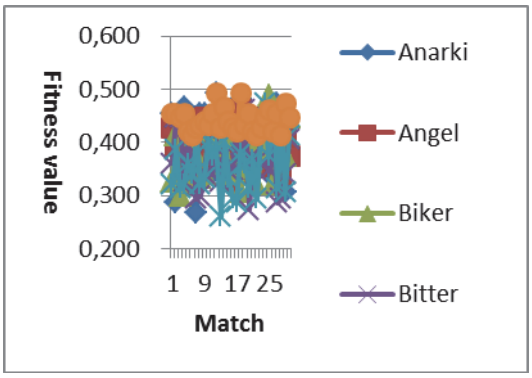

Figure 1: The turnament results.

5 CONCLUSIONS

In a first-person shooter, it is difficult to tune an agent's behavior since there are a large number of factors that affect the setting for a character. The proposed approach develops an effective way to determine the parameters for an intelligent character's behavior in a complicated gaming environment. In the experiment, one of the most intricate modes of a game, Free for All, is used to verify the proposed method. The results have demonstrated that the proposed method is able to design a character capable of evolving and winning. The character is able to determine a suitable action and behavior during the game. Our proposed approach provides a game designer in designing an intelligent character with an effective method.

REFERENCES

Messerschmidt, L., Engelbrecht, A. P., 2004. Learning to play games using a PSO-based competitive learning approach, *IEEE Transactions on Evolutionary Computation* 8(3), 280-288.

Ponsen, M., Spronck, P., Héctor, Muñoz-Avila, Aha, D. W., 2007. Knowledge acquisition for adaptive game AI, *Science of Computer Programming*, 67:59–75.

Shyr W. J., 2008. Introduction and Comparison of Three Evolutionary-Based Intelligent Algorithms for Optimal Design, *Convergence and Hybrid Information Technology, 2008.* ICCIT '08, 879 - 884.

Sipper, M., Azaria, Y., Hauptman A., Shichel, Y. 2007. Designing an evolutionary strategizing machine for game playing and beyond. *IEEE Transactions on Systems, Man, and Cybernetics – Part C: Applications and Reviews* 37(4)583-593.

Smith, M., Lee-Urban, S., Héctor Muñoz-Avila, 2007. RETALIATE: Learning winning policies in first-person shooter games, *American Association for Artificial Intelligence*, http://www.aaai.org

Tsai, C.-T., Liaw, C., Huang, H.-C., Ko, C.-H., 2011. An Evolutionary Strategy for a Computer Team Game, *Computational Intelligence*, 27(2), 218-234.

Zanetti, S., El Rhalibi, A., 2004. Machine learning techniques for FPS in Q3, *ACM '04*, Singapore.

Zarozinski, M., 2002. An open source fuzzy logic library, *AI Game Programming Wisdom*, Charles River Media Press, 90-101.