LEARNABLE BUDDY: LEARNABLE SUPPORTIVE AI IN COMMERCIAL MMORPG

Theppatorn Rhujittawiwat and Vishnu Kotrajaras
Department of Computer Engineering
Chulalongkorn University, Bangkok, Thailand
E-mail: g49trh@cp.eng.chula.ac.th, vishnu@cp.eng.chula.ac.th

KEYWORDS
Artificial Intelligence, Genetic Algorithm, Massively-Multiplayer Online Game.

ABSTRACT

In commercial massively-multiplayer online role-playing games (MMORPG), players usually play in a populated environments with simple non-player game characters. These non-player characters have fix behaviour. They cannot learn from what they experience in the game. However, MMORPG environments are believed to be greatly suitable for training AI, with plenty of players to provide tremendous amount of feedback, and persistent worlds to provide learning environments. This paper presents an experiment to find out the potential of MMORPG environments for fast learning evolutionary AI. The genetic algorithm is chosen as our learning method to train a non-player character to assist real players. We use a game server emulator and custom game clients to simulate and run a commercial MMORPG. Clients are divided into two groups, real players and "helpers". The results show that helpers can learn to assist real players effectively in small amount of time. This confirms that evolutionary learning can be used to provide efficient learning in commercial MMORPG. It also verifies that MMORPG provide great platforms for research in evolutionary learning.

INTRODUCTION

Recent game AI research and developments in online games are mostly focused on player opponent AI. Seu simulated and tested the system for evolving distribution, physical parameters, and behavior of monsters in game (Seu et al 2004). They found that monsters special qualities could be evolved according to their environments by using GA technique. Group movement was expressed by the flocking algorithm. However, actual learning was restricted to animal behaviour such as looking for food. Also, the length of time spent before monsters displayed satisfactory intelligent behaviour was not discussed. Spronck proposed a novel technique called "Dynamic Scripting" (Spronck et al 2004). Dynamic scripting used an adaptive rulebase for the generation of intelligent opponent AIs on the fly. In his experiment, a module for the commercial game NEVERWINTER NIGHTS (NWN; 2002) was created. A group of agents using dynamic script were pitted against various groups of pre-coded opponents. The results showed that dynamic scripting succeeds in providing clever AI in an acceptable period of time (around 50 battles needed for fighting with well coded opponents). However, a predefined rulebase was needed in this technique, meaning the actual time when learning from scratch was longer. Furthermore, although an agent learned using information from other agents in its team, one agent could only learn for itself at one time. A genetic algorithm was later used to create a rulebase for dynamic scripting (Spronck et al 2005). However, the work was carried out as an offline learning and learning time from scratch was not discussed. Stanley introduced the real-time NeuroEvolution of Augmenting Topologies (rtNEAT) (Stanley et al 2002). This is a learning method that was extended from NeuroEvolution of Augmenting Topologies for evolving increasingly complex artificial neural networks in real time, as a game is being played. The rtNEAT method allows agents to improve play style during the game. He demonstrated a new genre of games in which a player trains an agent team to compete with another player's team in NeuroEvoling Robotic Operatives (NERO) game (Stanley et al 2005). However, the nature of NERO implies that only one player can be training agents at one time.

MMORPG provides a very different environment and gameplay compared to other kinds of games.

With a massive number of players, these players can act as trainers for an evolving agent. Also, players spend more time playing MMORPG than other genres of games, and persistent world is used as a setting. This means MMORPG is likely to be a great environment for fast learning, even though we may use a slow learning method such as a GA. This paper presents the result of an experiment that evolves a player's helper in a commercial MMORPG game using a genetic algorithm. We call our player assistant a "Learnable Buddy".

Our learnable buddy technique has been tested by using the MMORPG server emulator of eAthena and custom client of OpenKore. eAthena is an open-source project, emulating a Ragnarok Online Server. It is written in C. Using its server emulator, a game server can be simulated and studied. OpenKore is an advanced bot for Ragnarok Online. It is free, open-source and cross-platform. In real MMORPG, many human players play in the same game server. We simulate human players by using OpenKore. Learnable buddy also makes good use of OpenKore. By modifying OpenKore code, we build AI-control units that are able to learn to improve their behavior.

Learnable Buddy

Learnable Buddy uses a genetic algorithm to set its configuration, which is a bot script. By evolving the chromosome of our population bots, our bots are able to perform various behaviors. The system consists of the following components.

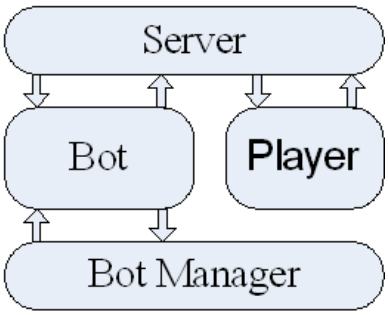

Figure 1: Learnable Buddy system overview.

1. Server: The game server sends game state information to every client and receives commands from clients. It keeps updating game state.
2. Player: All players are online, each can give us feedback.
3. Bot: Our bot is a supportive AI that travels along with a player. That player is a master and the bot is a slave. Bot systems have already been in use in various commercial games, such as the homunculus system in Ragnarok Online (RO; 2006). In Raknarok Online, players who play the alchemist or the biochemist can get a homunculus. The homunculus system surpasses other commercial MMORPG bots such as Guildwars's pet (Guildwars; 2006) because players are able to manually rewrite the bot's AI script. In this study, instead of using monsters as bots, we used player's character class as our supportive AI because a player character can perform more varying kinds of behavior. OpenKore was used to control each supportive AI. OpenKore was modified to send information and receive commands from the bot manager.
4. Bot manager: A module was written in Java. This module receives information from each bot, then determines their fitness and replaces low fitness bots with new ones. The detail is described below.

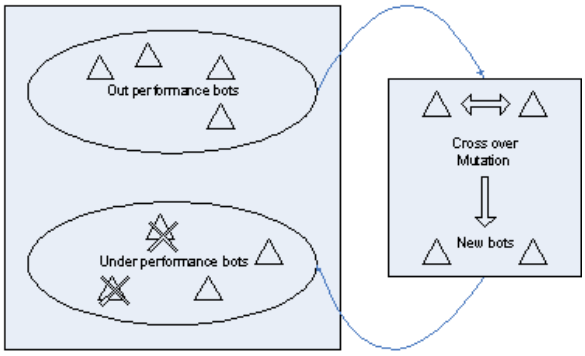

Figure 2: The replacement cycle

A bot plays using the first script it receives from the bot manager for a fixed period of time. Then, the bot manager will determine the fitness of each script. For this study, we use a fixed fitness equation. The fitness is calculated based on experience points a bot receives and the number of times that bot dies during the period. The value of fitness F, for bot b, is formally defined as:

$$F(b) = \frac{botEXPperHour(b)}{botDeadCount(b)^2}$$

The experience points that a master or its slave bot gain from any action will be divided in half. The bot receives the same amount of experience points

as its master. New chromosome generation is similar to regular GA techniques. First, good parents are chosen. Half of the bot population, whose with high fitness, are selected to produce offsprings that replace the half with lower fitness result. Each couple will perform a crossover, obtaining two new chromosomes. After that, new chromosomes will go through mutation. After a new chromosome is generated, the bot manager will read its attributes, transforming the attributes into a script, and replace a poorly performed bot with the new script.

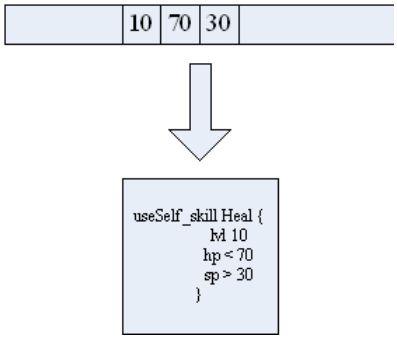

Figure 3: Example of chromosome to script translation

The openKore main script consists of 2 formats. The first format is of the form:

<configuration key> <value>

This format is used for a simple task. For example, in order to specify whether our OpenKore bot automatically attacks monsters, we use the following script:

attackAuto 1

where the proper value for this configuration is 0 (false) or 1 (true).

The second format is of the form:

<configuration key> <value> {
<attribute1> <value1>
<attribute2> <value2>
}

This format is called "block" format. It is used for a complicated task. In figure 3, OpenKore will use level 10 Heal skill on itself when its hp is less than 70% and its sp is greater than 30%. With this configuration structure, it is quite straightforward to translate between a script and its corresponding chromosome.

After new scripts are generated from the chromosomes of offsprings, half of the learnable buddies that used to have lower fitness results will reload new scripts and continue playing the game for another fixed period of time before repeating this cycle. The cycle can be done fast enough not disrupt the game play.

THE EXPERIMENTS

We assessed the performance of parties of two characters. We set up a private server using eAthena server emulator. Each party had the same members consisting of the following character.

1. Knight: The knights are bots that represent the real players who play a game. In this study, we used controlled experimental environments such that every player shared the same play style. All knights were implemented with the same script. This allowed learnable buddies to share their knowledge and learn together in a consistent way. The knights always attack the nearest monster that no one attacks. If a knight's health is reduced to half, it will rest until its health is fully recovered.
2. Priest: All priests are controlled by our learnable buddy technique. They will try to learn and adapt themselves to best serve their master. The priests support the knights with healing and buffing. Their behavior follows the script that they receive from the bot manager.

Testing was initiated using 16 pairs of knights and priests. Every party played in the same map that has only one kind of monster. The time cycle that we used for our fixed period was 30 minutes. Having a shorter cycle would affect the accuracy of the fitness result because the number of enemies faced might be too small and the fitness function might not show its effect because of that. On the other hand, our test platform could not run more than 30 minutes without a bot failing due to too much load the system had to handle. Therefore, the cycle of 30 minutes was our best choice.

To quantify the performance of each learnable buddy, after each time cycle, we calculated the

fitness for each learnable buddy by the function from our previous section and replaced poorly performed bots with new ones. We ran 3 tests, each test ran for 50 generations of learnable buddy. The results of these experiments are presented in the next section.

RESULT

Figure 4 shows fitness mean of bots. A solid line represents fitness mean of each generation. It can be observed that, from the beginning until around the fifteenth generation our bots' fitness mean rapidly increases. The fitness does not vary much after that. Figure 5 shows the result of figure 4 after smoothness adjustment, using polynomial degree 5 trend line.

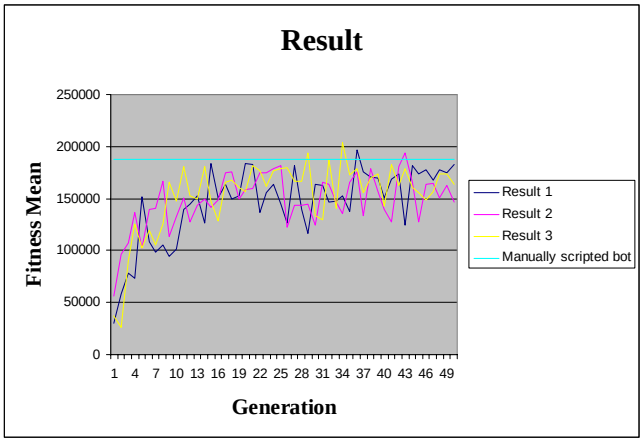

Figure 4: Resulting graph of learnable buddy, using three test runs.

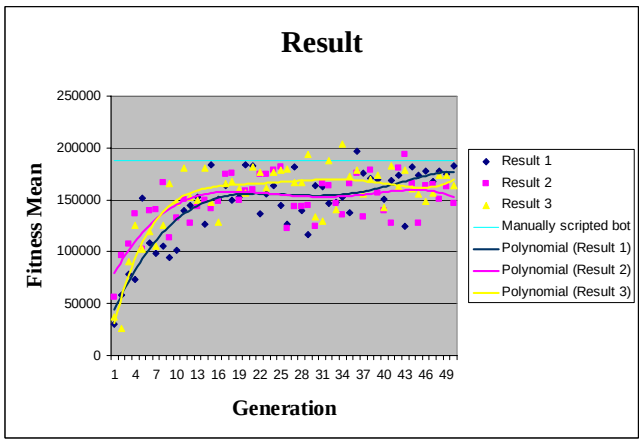

Figure 5: Resulting graph of learnable buddy after smoothness adjustment using polynomial degree 5 trend line.

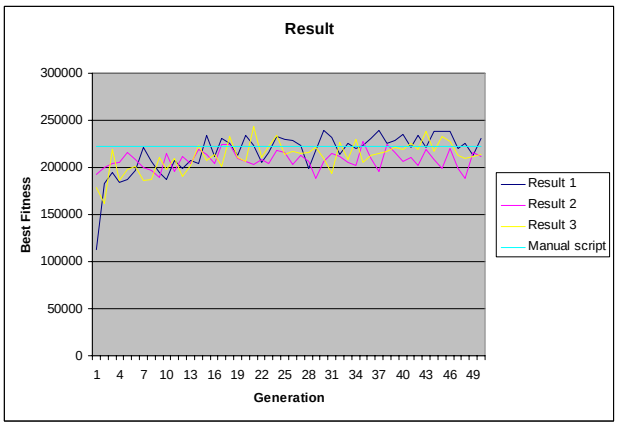

Figure 6: Resulting graph of best fitness in each generation competing against best fitness of our manully scripted bot.

We observed and compared the 15th generation of learnable buddies with manually scripted supportive characters configured by an experienced game player. The mean fitness of our bots came close to the mean fitness of the manually scripted bot. Not all of our best bots in the 15th generation could beat the manually scripted bot's best score. But observing the results for their future generations suggested that our best bots could compete well with the manually scripted bot (see figure 6). From the result, we believe that, in order to help one master with a task, our learnable buddies can improve themselves to their proper performance in around fifteen generations or 7.5 hours of playing. The survey from America Online shows that teenage players spend 7.4 hours per week on average playing online games (AOL 2004). Therefore our 7.5 hours figure is significant. It means one task can be learned in just a week for the same group of real players. Most MMORPGs plan to let players play for several months or maybe a year, therefore one week is considered to be very efficient. It can even be improved further. A bot can be kept running for 24 hours by assigning it to another player. Therefore, fast learning for a task can be achieved.

CONCLUSION AND FUTURE WORK

In this paper we investigated whether evolutionary-learning can provide fast online adaptation of player supportive AI in commercial MMORPG. From our experimental results, we conclude that genetic algorithm is fast and effective enough for commercial MMORPG. The

original game does not need to be adjusted in any way. Different genes can be used for different tasks and players can switch between tasks to allow more suitable behaviour at each situation.

Currently, our bot manager only supports fixed fitness function given by game developers. That means, only common tasks can be learned. To allow supporting AI to be able to learn more tasks or even improve upon old tasks, especially ones specific to events or groups of players, players must be able to craft their own fitness function through an intuitive interface. We also plan to experiment with genetic programming, which allows builds-up of complex behaviour. One of our research goals is to be able to categorize player behavior while playing. This will permit learnable buddies to automatically switch to the script that best fits the situation, thus adding more sense of realism.

ACKNOWLEDGMENT

This research is sponsored by Ratchadaphiseksomphot Endowment Fund, Chulalongkorn University.

REFERENCES

Kenneth O. Stanley, Bobby D. Bryant and Risto Miikkulainen. 2005. “Evoling Neural Network Agents in the NERO Video Game.” In Proceeding of IEEE 2005 Symposium on Computational Intelligence and Games (CIG’05).

Kenneth O. Stanley and Risto Miikkulainen. 2002. “Efficient Reinforcemant Learning through Evoling Neural Network Topologies.” In Proceedings of Genetic and Evolutionary Computation Conference (GECCO-2002).

Kenneth O. Stanley and Risto Miikkulainen. 2002. “Evolving Neural Networks through Augmenting Topologies.” The Massachusetts Institute of Technology Press Journals, Evolutionary Computation 10(2). 99-127

Jai Hyun Seu, Byung-Keum Song and Heung Shik Kim. 2004. “Simulation of Artificial Life Model in Game Space.” Artificial Intelligence and Simulation, 13th International Conference on AI, Simulation, and Planning in High Autonomy Systems. 179-187

Marc J.V. Ponsen, Héctor Muñoz-Avila, Pieter Spronck, and David W. Aha. 2005. “Automatically Acquiring Adaptive Real-Time Strategy Game Opponents Using Evolutionary Learning.” Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, pp. 1535-1540. AAAI Press, Menlo Park, CA.

Pieter Spronck, Ida Sprinkhuizen-Kuyper and Eric Postma. 2004. “Online Adaptation of Game Opponent AI with Dynamic Scripting.” International Journal of Intelligent Games and Simulation, Vol 3 No 1, 45-53

America Online (2004)
http://www.aol.com

eAthena (2006). Ragnarok Online Server Emulator
http://www.eathena.deltaanime.net

GuildWars (2006)
http://www.guildwars.com

Neuro-Evolving Robotic Operatives (2006)
http://nerogame.org

NEVERWINTER NIGHTS (2002)
http://nwn.bioware.com

OpenKore (2006). Ragnarok Online Bot
http://www.openkore.com

Ragnarok Online (2006)
http://www.ragnarokonline.com