國立交通大學

資訊科學與工程研究所

碩 士 論 文

利 用 計 算 智 能 處 理 遊 戲 團 隊 平 衡

Game Team Balancing by Using Computational Intelligence

研 究 生：方士偉

指導教授：黃世強　教授

中 華 民 國 一 百 年 十 一 月

利用計算智能處理遊戲團隊平衡

Game Team Balancing by Using Computational Intelligence

研　究　生：方士偉　　　　Student　：Shih-Wei Fang

指 導 教 授：黃世強　　　　Advisor　：Sai-Keung Wong

國　立　交　通　大　學

資 訊 科 學 與 工 程 研 究 所

碩　士　論　文

A Thesis

Submitted to Institute of Computer Science and Engineering

College of Computer Science

National Chiao Tung University

in Partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

October 2011

Hsinchu, Taiwan, Republic of China

中　華　民　國　一　百　年　十　月

利用計算智能處理遊戲團隊平衡

研究生：方士偉　　　指導教授：黃世強　教授

國 立 交 通 大 學

資 訊 科 學 與 工 程 研 究 所

摘要

在本篇論文裡，我們提出了一個新的團隊能力平衡系統 (TABS)，以協助遊戲設計師評量一個角色扮演團隊戰鬥類型的遊戲內，任兩個團隊的能力設定是否平衡。TABS 利用基因演算法或粒子群優化，來訓練人工神經網路控制器。訓練成果最傑出的控制器會被選擇並應用於團隊平衡度的評估。另外，我們也提出了教練訓練法這種訓練模式，它能公平地訓練控制器。為了加速訓練的程序，我們運用了多執行緒技術，讓控制器能在數個獨立的遊戲空間裡平行進行訓練。在個案研究中，我們把 TABS 應用到我們自行設計的 MagePowerCraft 遊戲來進行驗證與實驗。而實驗結果顯示我們的系統效能頗令人滿意。

Game Team Balancing by Using Computational Intelligence

Student: Shih-Wei Fang Advisor: Sai-Keung Wong

National Chiao Tung University

Institute of Computer Science and Engineering

Abstract

In this thesis, we propose a novel Team Ability Balancing System (TABS) to assist game designers to evaluate whether the team ability settings of any two teams are balanced in a role-playing combating game. TABS uses artificial neural network controllers which are trained by either genetic algorithm or particle swarm optimization. The best-trained controllers are chosen and applied for the team balance evaluation. Additionally, we also propose the Train-by-Coaches scheme which is useful for training controllers in a fair manner. In order to speed up the training process, we apply the multi-threading techniques to train the controllers in several independent game spaces in parallel. In a case study, we apply TABS to the in-house designed game, MagePowerCraft, for validations and experiments. Experimental results show that the performance of our system is quite satisfactory.

Acknowledgements

This thesis would not have been possible without the assistance of many people whose contributions that I gratefully acknowledge.

I owe my deepest gratitude to my advisor, Dr. Sai-Keung Wong, for his insightful suggestions, helpful assistances and also his positive encouragements from the preliminary to the concluding level of this thesis.

I would like to show my gratitude to my dearest lab mates, who had been supporting me both technically and spiritually. It is an honor for me to have such great lab mates, I really do appreciate the happy moments in our lab.

If it is not the love and support that I had received from my parents and my lovely sister, this thesis would never accomplish, I would like to show my family how much I love them.

For the 3D models that I had used in my thesis, I would like to thank the people who made them and contributed them to the society, thanks for their selfless dedications.

Lastly, I offer my regards and blessings to all of those who supported me in any respect during the completion of the thesis.

Shih-Wei Fang

October, 2011

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Artificial intelligence techniques have been widely developed for computer games in different aspects. Despite their different purposes in the development of computer games, the common goal is to make the games more enjoyable and fun.

1.1 Motivation

In order to uphold fun and fairness in a multiplayer role-playing video-game, keeping the power of abilities balanced among the characters of different classes, races, or alliances is an important issue. The players' gaming experiences may be affected by the fairness in these games. If the game is well balanced, then the abilities of a character/team should not be more powerful than the other characters/teams. Game balance is strongly demanded in online games which include Player Killing (PK) systems or in the games which divide their players into two or more alliances that are hostile to each other. And to achieve such demand, it could involve tuning hundreds or even more than thousands of parameters and settings. For some games, this may be a never-ending tuning cycle. It may consume a lot of time, human resources and money.

Currently, the typical solution to evaluate and achieve game balance is to run a lot of test series which are played by human or scripted artificial intelligence (AI). Some other game companies might perform an actuarial study about the abilities of all the character classes or the teams to check game balance. Using scripted AI seems to be a rather economic method; however, the results of the tests might be biased due to the changeless routine actions defined by the scripts.

In this thesis, we present a novel *Team Ability Balancing System* (TABS), which can assist game designers in evaluating the situation of team balance automatically for a role-playing team combating game.

1.2 Overview

TABS uses artificial neural network (ANN) controllers which are trained by either using the model of genetic algorithm or particle swarm optimization. During the training session, the controllers of each character class learn the ways to assist team-mates, attack opponents and avoid damages. We also propose a training scheme, Train-by-Coaches, for training the controllers efficiently. Moreover, since both genetic algorithm and particle swarm optimization are population-based heuristics. The training session is the most time-consuming process, and it is the bottleneck in TABS. To ease such problem, we propose to apply the multi-threading technique for parallel training.

After the training session, the best controllers are used for controlling the characters of two teams and fight with each other. Meanwhile, the performance scores and gaming statistics of the two teams are recorded. In order to obtain reliable data for analysis, we repeatedly perform game tests for a number of times. Based on the performance scores, game designers can adjust the parameters of the abilities of the character classes accordingly and run tests with their new settings by TABS so as to achieve game balance.

1.3 Contribution

The major contributions of this thesis are described as follows.

1. A novel Team Ability Balancing System (TABS) is proposed. TABS can help the game designers tuning the settings of game character classes of a role-playing com-

bating game efficiently by providing the balance information of current setting.

2. This is the first attempt of training artificial neural network controllers to help evaluate the balance situation of game character abilities.

3. Two training models, genetic algorithm and particle swarm optimization, are implemented for training controllers in our system.

4. A Train-by-Coaches training scheme is proposed to train the controllers in a fair manner is proposed.

5. Artificial neural network controllers are trained in parallel by multi-threading technique in serveral independent game spaces.

6. An automatic training process with premature convergence check and handling.

1.4 Organization

The remaining chapters are organized as follows. Chapter 2 reports the related work about artificial intelligence in games, and also the researches on genetic algorithm, particle swarm optimization and artificial neural network. The controller and training models that applied in this thesis is described in Chapter 3. Chapter 4 presents our system, TABS, and the details of implementations. A case study of applying TABS with a in-house designed game is given in Chapter 5 which also includes validations, experiments and their results. Some discussions are shown in Chapter 6. Finally, Chapter 7 gives the conclusion and future work.

Chapter 2

Related Work

Artificial intelligence techniques are not only applied for solving the problems of path finding [FMD02, GMS03], controlling the non-player characters with a variety of reactions to players in intelligent and challenging ways, but also learning the behaviours of players [TBS04]. Some techniques may be involved with dynamically adjusting the game difficulty for game balancing [HC04] or even changing game parameters via online learning algorithms [FP10].

Genetic algorithm (GA) is one of the evolutionary algorithms inspired by the process of natural evolution [G^+89, Hol92]. It gives solutions for optimizing problems by applying the techniques, such as, inheritance, mutation, selection and crossover. Particle swarm optimization (PSO) was introduced by Kennedy and Eberhart [KE95]. The development of PSO was inspired by the elegant motion of a flock of birds searching for food. PSO can be adopted to optimize continuous functions by Kennedy [Ken97]. Shi and Eberhart [SE98] proposed a method for PSO parameter selection. PSO has also been adapted to various applications [ES01], such as ANN controllers evolving, system design, pattern recognition and scheduling. Angeline [Ang98] investigated both the philosophical and performance differences of PSO and evolutionary algorithms. Eberhart and Shi [ES98] compared PSO and GA in details.

Leigh et al. [LSL08] tried to balance gaming environment by using coevolutionary algorithms. There is little work devoted for game balance between teams in role-playing video games. However, there are plenty of works about games or gaming mechanisms which are developed based on ANN, GA or PSO. Branke [Bra95] has seen applications in game development. Cole et al. [CLM04] presented methods for tuning first-person shooter bots by applying genetic algorithms. The video game *NERO*, which was developed by Kenneth et al. [SBM05], is one of the innovative examples. The agents in the game are capable of learning online while the game is being played. The skills of the agents are evolved gradually. Olesen et al. [OYH08] presented real-time difficulty adjustment and Thrun [Thr95] used ANN to play chess. Riechmann [Rie01] connects the theory of genetic algorithm to evolutionary game theory. Revello and McCartney [RM02] applied genetic algorithm to war games which contain uncertainty. Cardamone et al. presented controllers for car racing games with neuroevolution [CLL09]. Wong [Won08] utilized backpropagation neural network for personalised difficulty adjustment in a game system.

Chapter 3

Controller and Training Models

3.1 Artificial Neural Network Controller

Artificial Neural Network (ANN) is a computational model inspired by the operation of biological neural network. An ANN consists of a group of connected artificial neurons. Due to the modeling ability of nonlinear statistical data, ANNs are applied in a variety of fields as tools which model the complex relationships between inputs and outputs. Figure 3.1 shows the structure of a typical feedforward ANN with one hidden layer and the propagation flow.

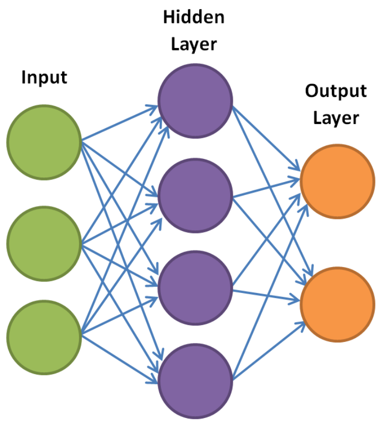


Figure 3.1: The structure of a feedforward ANN.

Each artificial neuron will sum up all the weighted inputs and the bias, and then the summation will be filtered through an activation function which determines the output of the neuron. The structure of an artificial neuron is shown in Figure 3.2, where x_1 to x_n are the n inputs to the neuron, w_1 to w_n are the weights for each input, w_b is the weight for the bias, $+$ stands for summation, and f is the activation function.

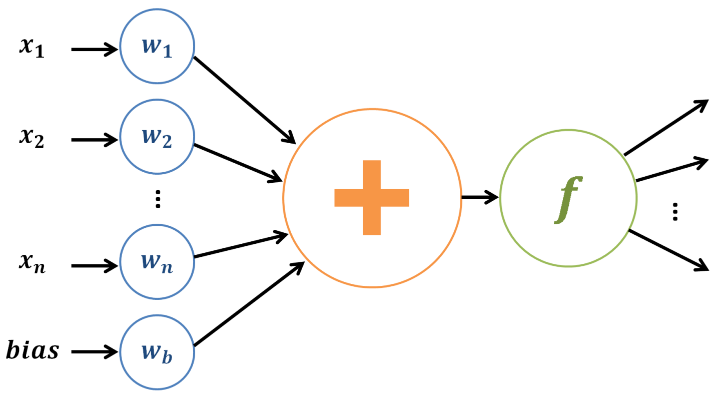


Figure 3.2: The structure of an artificial neuron.

To contol a game character, we feed sufficient gaming information as inputs to an ANN, and the outputs of the ANN are corresponding to the available actions of the game character. For simplicity, all ANNs in TABS have identical hidden layer structure, which consist of one layer with six neurons. Sigmoid function $P(t)$ is adopted as the activation function, as shown in Equation 3.1.

$$P(t) = \frac{1}{1+e^{-t}} \tag{3.1}$$

3.2 Modeling Using Genetic Algorithm

As one of the evolutionary algorithm, genetic algorithm (GA) is inspired by the process of natural evolution. GA gives solutions for optimization and search problems by applying the techniques which mimic natural evolution in the genetic level, such as inheritance, selection, mutation and crossover.

In GA, a chromosome (or a string) represents one of the candidate solutions of the problem. And a chromosome is usually encoded by a set of genes. While combining GA with ANN controller, the problem would become as trying to evolve the best controller after a number of generations. That is to find the best combination of the weights in the ANN. Therefore, we need to encode the set of weights and treat them as genes for evolving the ANN controllers gradually. Initially, all the weights of the ANNs of the whole population are randomly generated. During the process of evolution of each generation, a fitness value will be calculated and assigned to every controller by a predefined fitness function, which is used to measure the quality of the controllers. The controllers with the highest fitness values are regarded as the elites of the population. The next generation of the population is created by the process of inheritance, selection, mutation and crossover. With a well-designed fitness function, we expect to get better and better solutions through the generations of evolving.

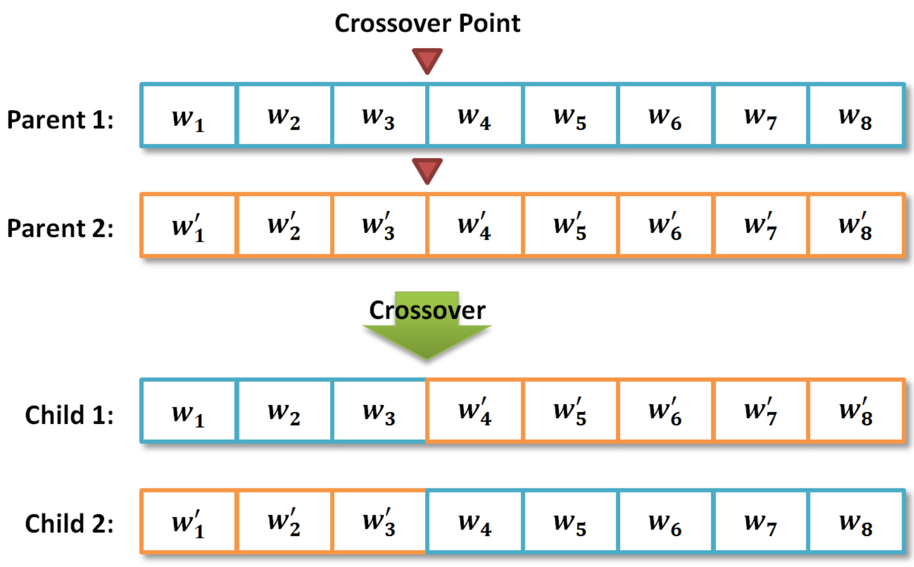

Crossover Point
Parent 1:
w_1 w_2 w_3 w_4 w_5 w_6 w_7 w_8
Parent 2:
w'_1 w'_2 w'_3 w'_4 w'_5 w'_6 w'_7 w'_8
Crossover
Child 1:
w_1 w_2 w_3 w'_4 w'_5 w'_6 w'_7 w'_8
Child 2:
w'_1 w'_2 w'_3 w_4 w_5 w_6 w_7 w_8

Figure 3.3: A crossover operation generating two child chromosomes from two parent chromosomes. The genes (i.e. the weights) of the two parent chromosomes beyond the crossover point are swapped.

In order to evaluate how good the controllers in TABS are, fitness functions are usually heavily depended on the rules of games. For the genetic operators in TABS, all operators are directly applied for modifying the genes in TABS. we adopt the roulette-wheel sampling for selection, which is also known as fitness proportionate selection. TABS applies one-point crossover. When crossover is performed for two chromosomes, a crossover point is randomly selected and then all the genes beyond the crossover point

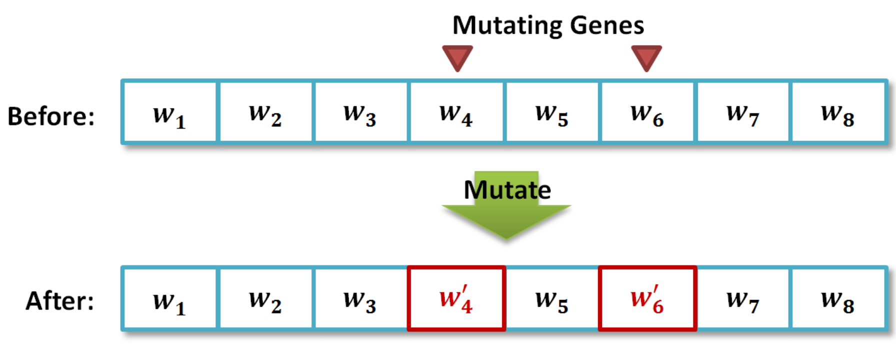

Mutating Genes
Before:
w_1 w_2 w_3 w_4 w_5 w_6 w_7 w_8
Mutate
After:
w_1 w_2 w_3 w'_4 w_5 w'_6 w_7 w_8

Figure 3.4: Each gene has a probability to mutate. In TABS, the genes are mutated by perturbing the weights by a random amount.

of the two chromosomes are swapped. If mutation is performed, a subset of weights of the gene is selected by the mutation rate and each weight is perturbed by a random amount. Figure 3.3 and 3.4 show the operation of crossover and mutation respectively. Table 3.1 shows the detail GA settings of TABS.

Population Size	100	Crossover Rate	0.5	Mutation Rate	0.05
Pertubation Value	-0.3 $\sim$ 0.3	Number of Elites	2	Copies of Elite	1

Table 3.1: Detail settings of GA in TABS.

3.3 Modeling Using Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a computational method which gives solutions for optimization and search problems iteratively. There are several variations of PSO with different topologies. In TABS, we implemented the PSO algorithm based on the one which is proposed by Venter and Sobieski [Ven02]. In PSO, the individuals of the swarm are interpreted as particles moving in the search-space of the problem. The position of each particle represents a candidate solution of the problem. Initially, the positions and velocities of all the n particles are randomly generated over the entire search-space. The positions of the particles are updated by their velocities after each iteration (move) for refining the candidate solutions. Each particle i has a position x_i and a velocity v_i. The velocity of a particle i is given by:

$$v_i^{t+1} = wv_i^t + c_1 r_1 (p_i - x_i^t) + c_2 r_2 (p_g^t - x_i^t), \tag{3.2}$$

where the superscripts $(t+1)$ and t denote the iteration steps, p_i is the best position which particle i has found so far, p_g^t is the global best position of all the particles at iteration t, and both r_1 and r_2 are the random values in $[0, 1]$; w, c_1 and c_2 are user-defined constant for controlling the behaviour and efficacy of the PSO method. w is the inertia factor, which control the impact of current motion. c_1 is the self confidence factor that regulate the effect of particle memory. And the influence of swarm knowledge is controlled by c_2, the swarm confidence factor. After the velocities of all the particles are updated, their positions are updated as shown in Equation 3.3 and as illustrated in Figure 3.5.

$$x_i^{t+1} = x_i^t + v_i^{t+1} \tag{3.3}$$

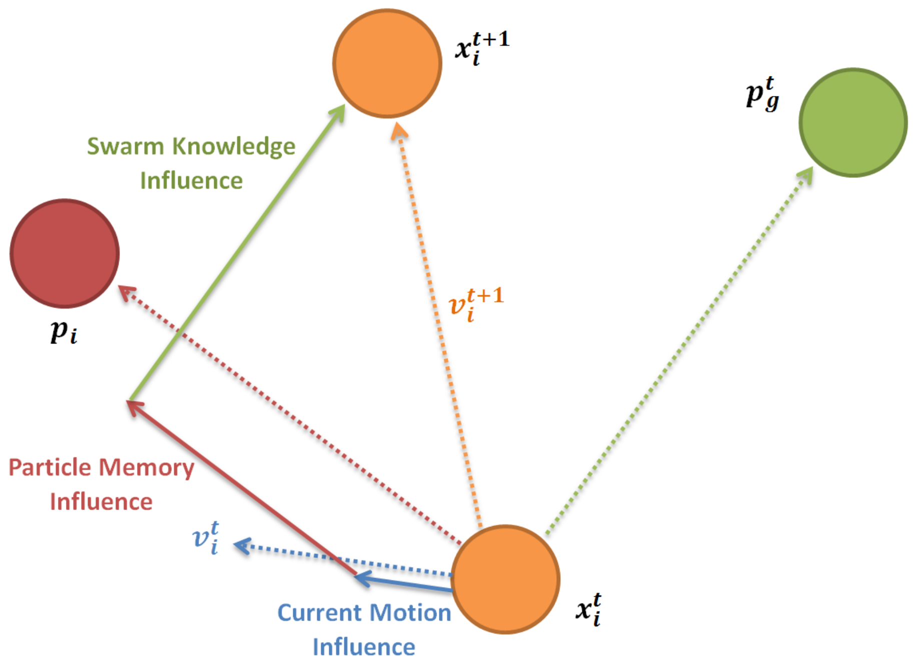


Figure 3.5: Velocity and position update of paritcle i in PSO.

Note that if a particle moves outside of the search-space, the particle is set back to its nearest side constraint and its velocity is set to zero.

When combining PSO with ANN controller, we want to develop the controller with the best combination of the weights in the ANN as GA that we had mentioned in Section 3.2. In this case, the search space would be a λ-dimensional space, where λ is the total number of weights of an ANN. The fitness function for evaluating ANN controllers is the same as the one we used in GA. Although it would be an interesting topic which worth further research and investigation for tuning the factors w, c_1 and c_2, it is outside of the scope of this thesis. In TABS, a fast convergence rate can be achieved by setting w, c_1, and c_2 as 0.5, 1.5 and 1.5, respectively.

Chapter 4

Team Ability Balancing System (TABS)

Team Ability Balancing System (TABS) is a system designed by us, used for evaluating the balance situation between teams of a role-playing team combating game. We use the ANN controllers trained by TABS to control the behaviours of game characters. These controllers will then fight in a series of team battle games and TABS will evaluate the balance situation between the teams accordingly by the results of the games. Currently, TABS can handle two teams fighting with each other and each team can have up to a maximum of three characters.

4.1 Workflow of TABS

The workflow of TABS is depicted in Figure 4.1. We describe the steps of the workflow in details as follows.

Setup GA/PSO, ANNs and Game Spaces

During the preprocessing stage, TABS will setup either GA or PSO based on user's selection. Assume that we have a number of characters divided into two teams which are subjected for being evaluated their ability power between them. For each character, we build up its own training manager and create a population of n candidate controllers. A number of game spaces are prepared for parallel training in the later-on steps.

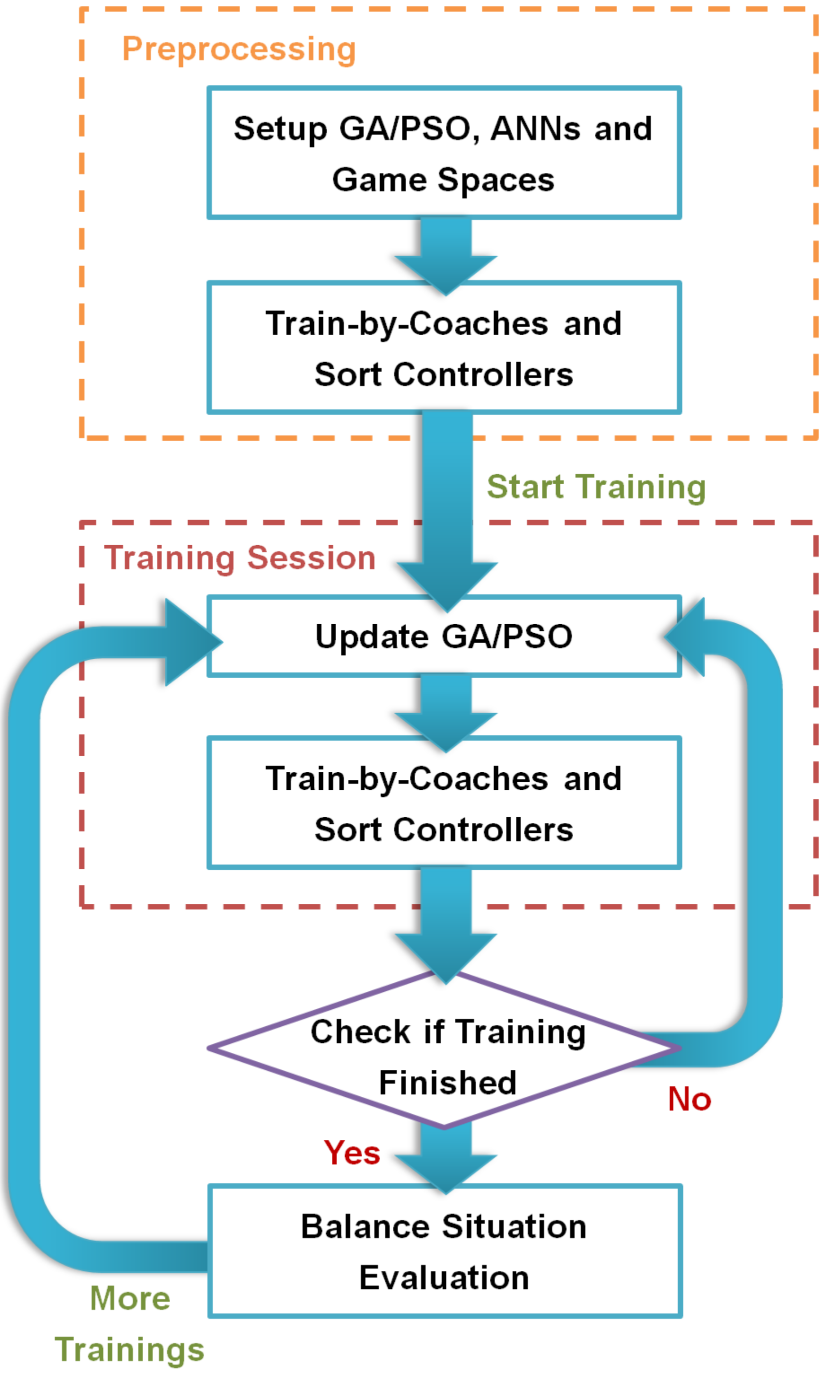

Preprocessing
Setup GA/PSO, ANNs and Game Spaces
Train-by-Coaches and Sort Controllers
Start Training
Training Session
Update GA/PSO
Train-by-Coaches and Sort Controllers
Check if Training Finished
No
Yes
Balance Situation Evaluation
More Trainings

Figure 4.1: The workflow of TABS.

Train-by-Coaches and Sort Controllers

The main goal of this step is to find out the best candidate controller of every character in the current state. Each candidate controller is arranged to train by a game simulation in a game space. After each game simulation is over, a fitness value is computed and assigned to the candidate controller according to the fitness function. Once all the candidate controllers had finish training, we may sort the controllers by their fitness values. The higher value represents the fitter (better) controller it is. Furthermore, in order to speed up the process of training, we train the candidate controllers in parallel by the help of multi-threading.

Update GA/PSO

The training manager of each game character handles the update of GA or PSO.

GA: According to the fitness values of the candidate controllers, a number of *elites* are chosen with their chromosomes copied and reserved to the next generation. Besides the elites, the rest population of the next generation is generated through the techniques of selection, crossover and mutation as stated in Section 3.2.

PSO: The current global best position p_g^t is equal to the (ordered) weights of the ANN of the best candidate controller. The best position ever found (i.e. p_i) of a particle is equal to the (ordered) weights of its corresponding ANN which has the highest fitness value so far. All particles are updated based on Equations 3.2 and 3.3.

Check if Training Finished

Once the maximum training generation/iteration is reached, or the termination criteria are met, then stop and exit the training session. Otherwise, go back to the training session and keep on training.

Balance Situation Evaluation

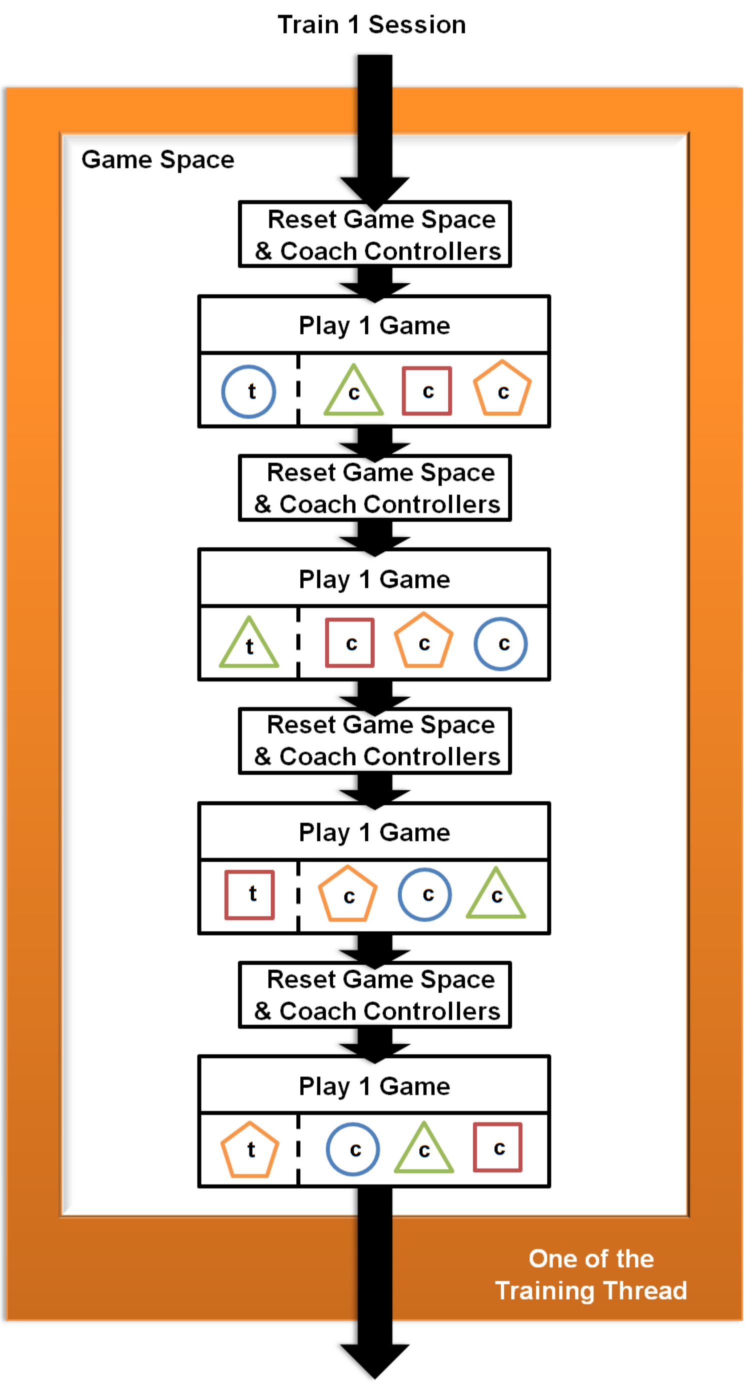
After the training session is over, we use the best-trained candidate controller of each game character to form teams and fight a series of battle games in teams and the gaming statistics are recorded. The balance situation of the two teams can be evaluated from the scores of the battle game series. With these statistics, TABS can also compute the performance scores of each individual character.

4.2 Train-by-Coaches

TABS trains the controllers by using the scheme of Train-by-Coaches. As TABS adopts GA and PSO for the training models, these two models are both unsupervised training models. In other words, the controllers need to be training according to their fitness values. We can tell that a controller is superior or poorer than one another by comparing their fitness values.

As fitness value is calculated by the fitness function and it is assigned to a controller based on the performance of the controller, one may find out that the fitness value of a controller is affected by the ability level of its teammates and the opponents. Imagine that we had two training controllers with the same ability level. And during the combat simulation, one of the controllers is arranged to fight with an opponent which is relatively weaker while the other faces a much tougher opponent. We might reasonably expect that the previous controller ought to obtain a higher fitness value than the latter controller. In this case, fitness value lost its credibility due to different standards of training. In fact, if the controllers were trained by different standards, a good controller might have lower fitness value than a bad candidate controller. As a consequence, the whole population will be extremely hard to improve themselves.

To overcome this problem, we propose the Train-by-Coaches training scheme which can make sure that all controllers are trained fairly. Assume that we have m characters with a population size of n candidate controllers for each character. In additional to the candidate controllers that are to being trained, we also create an extra controller, which



Train 1 Session
Game Space
Reset Game Space & Coach Controllers
Play 1 Game
t
c
c
c
Reset Game Space & Coach Controllers
Play 1 Game
t
c
c
c
Reset Game Space & Coach Controllers
Play 1 Game
t
c
c
c
Reset Game Space & Coach Controllers
Play 1 Game
t
c
c
c
One of the Training Thread

Figure 4.2: Training a set of four candidate controllers in a game space by the scheme of Train-by-Coaches. The four shapes: circle, triangle, square and pentagon represent the controllers of four characters. 't': candidate controller to be trained; 'c': coach controllers.

is the coach controller for each character. Initially, the ANN weights of these coach controllers are randomly generated as like the candidate controllers.

During the training process of a generation/iteration, every candidate controller of each character is trained by playing a game with the other coach controllers (not including its own coach controller) one by one. After all the candidate controllers of all characters have finished their games, we sort and find out the best candidate controllers of each character based on their fitness values. Then for each character, we make a copy of its best candidate controller. After that, we replace the coach controller of each character by the copy of the best candidate controller of the character. This makes the best candidate controller of a character in the current generation/iteration will be the coach controller of other characters during the next generation/iteration. The performance of the coach controller can therefore hopefully be improved after each generation/iteration gradually. Training the candidate controllers by the coach controllers also ensures that each candidate controller of the same character is trained by the controllers of the same level. In other words, the controllers of a character are all trained in a fair manner by the same standard. In this way, it is faster to train the candidate controllers with satisfactory performance. Figure 4.2 shows an example scheme of Train-by-Coaches for training a set of four candidate controllers in a game space.

4.3 Parallel Training

The training time of TABS is mainly dominated by the total time of game simulations which are held for training the candidate controllers. And the time needed for a game simulation is affected by the number of controllers involved in one simulation. Assume that we have m characters and with a population size of n candidate controllers for each character needed to be trained. A candidate controller is trained with $(m - 1)$ coach controllers at a time in one single game simulation. Therefore, m controllers are involved in one single simulation. We have $m \times n$ games needed to be simulated for the candidate controllers of all characters. As a result, the time needed for all simulations is affected by a factor of nm^2 for one generation/iteration. Moreover, the complexity of the game-logic will also affects the simulation time. That is because we need to execute the game-logic for every involving controller for every simulation time step. As Branke [Bra95] had concluded that adopting a multi-threading parallel approach for training/evolving controllers could alleviate the computation cost problem [CP98]. Therefore, one could see that parallel computation is highly demanded.

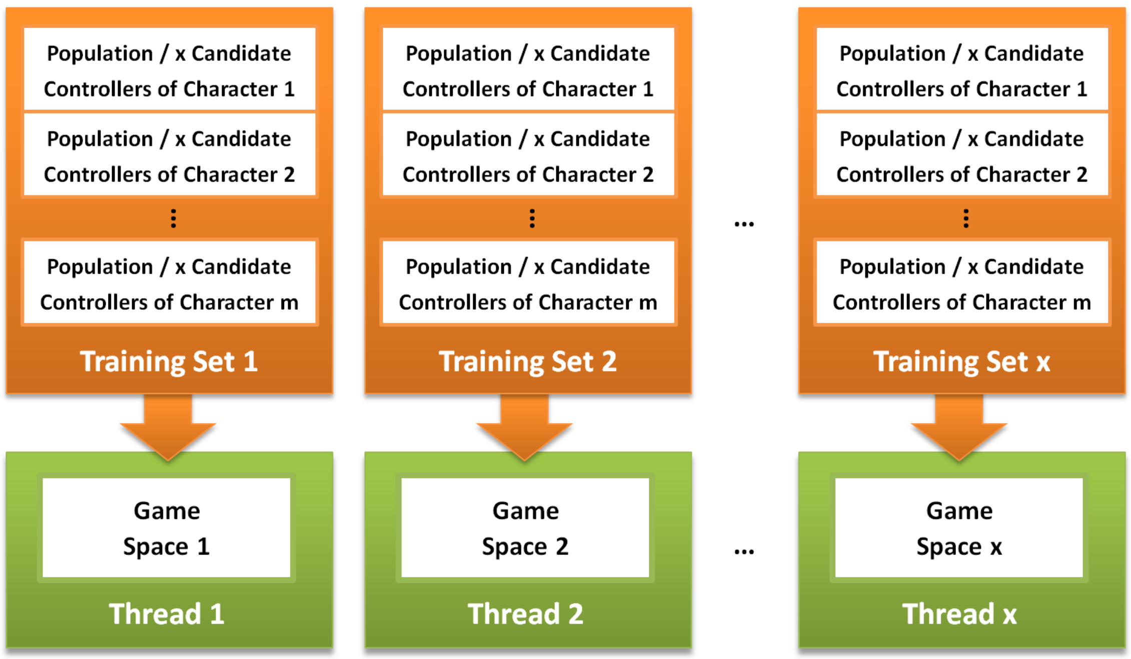

Population / x Candidate Controllers of Character 1
Population / x Candidate Controllers of Character 2
Population / x Candidate Controllers of Character m
Training Set 1
Training Set 2
Training Set x
Game Space 1
Game Space 2
Game Space x
Thread 1
Thread 2
Thread x

Figure 4.3: The architecture for parallel training.

In our approach, we create x threads on a multicore system. Each of the threads owns a game space which is created earlier in the preprocessing steps. Each game space has its own dynamic gaming data and the game-logic can be processed independently. The dynamic data include the positions and status of the characters. The static data, such as the 3D mesh data of models, can be shared by all the game spaces. All dynamic data of a game space need to be reset before a training session starts. For parallel training, we evenly divide each population of the m characters into x groups and assign them to the x threads as illustrated in Figure 4.3. In this way, we can perform training for the candidate controllers in parallel by using multi-threading technology on CPU. Usually a random number generator keeps its own state. To achieve thread safe for computing random numbers, we assign each game space a random number generator. We employ Mersenne Twister [MN98] for computing random numbers in TABS.

4.4 Automatic Training

TABS can train the ANN controllers automatically. After a number of generation/ iteration, TABS will start checking if all the termination criteria are met. The termination criteria are as follows:

T1. The growth of the best fitness values of the all populations converged.

T2. The current best fitness value of each character is not less than the 90% of the best fitness value of the last generation/iteration.

T3. Not in training extending state and no training extension is required.

TABS checks the growth convergence of the best fitness value by comparing the average fitness value of the recent w records, namely Avg_{Back} with the average fitness value of the w records before $dist_{Sampling}$ generations/iterations, Avg_{Front}, where w is the window size and $dist_{Sampling}$ is the sampling distance, as illustrated in Figure 4.4. If $|Avg_{Front} - Avg_{Back}|/Avg_{Front} < 5\%$, then TABS will claim that the growth of the best fitness value of such population is converged.

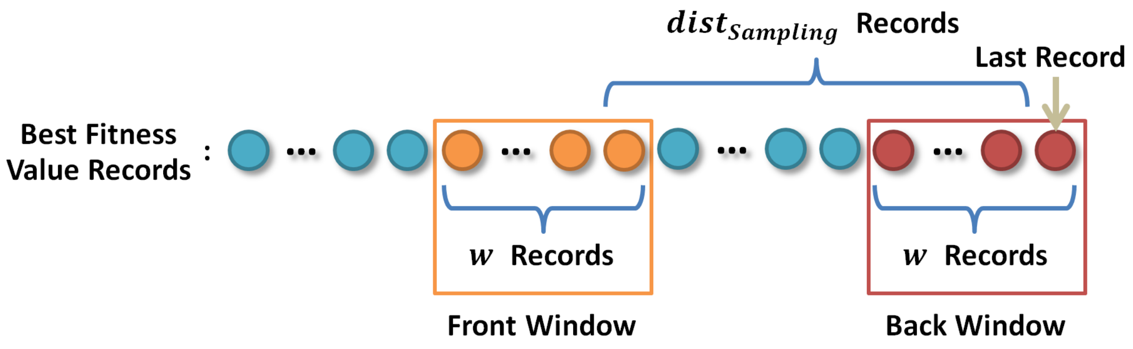


Figure 4.4: Convergence is checked by comparing the averages of the w records in the front window with the back window.

Once criteria T1 and T2 are met, TABS will further check the maturity of the current best controllers. Since in a combating game, whether a controller has learnt to cast all of its owning skills is rather important and also may regard as an indicator for the maturity of the controllers. Thus, if any skill of the current best controllers has never been cast, then TABS will perform a training extension and enter the training extending state. If such training extension had not been performed for two times already, TABS will be forced to keep on training all the candidate controllers for a number of generations/iterations predefined by the system user. Otherwise, we will assume the skills that have never been cast are useless for the characters and terminate the training session immediately.

Consider the case that a training extension is required and there are any skills that the best controller of a character has never cast them. If the training model is GA, then TABS will re-randomize all the genes of some chromosomes which have the lower fitness values. On the other hand, if the training model is PSO, TABS will re-randomize the current position x_i and the current velocity v_i of some particles i, which have the lower fitness values of the population. This partial re-randomization may do some help for the particles to escape away from the local optima as the Social Disasters Technique (SDT) introduced by Kureichick and colleagues [RN04] for avoiding the premature convergence to local optima.

4.5 Game Simulation

A game is simulated by following its rules or game logic in TABS. And in most cases, different games usually have different rules or game logic which make human players enjoy playing with. Despite the differences among different games, when evaluating the fairness of a game; TABS ignore all the factors that are affected by human-machine interaction or human error. These might include things such as the accuracy of aiming targets and pressing keys for certain actions (e.g. casting spells), or the legacy of human reaction to an event, etc. These factors affect the game play vary from person to person and lead to difficulties in analysing the ability power of the characters. In TABS, we assume that a controller can perform any available actions of the game accurately and immediately.

However, TABS needs to develop controllers that can be adapted to various game-plays of different games, in which the attributes of characters and the game rules differs from one another. To do so, some game-depended parts of TABS need to be custumised for each game. And among them, the in/output of ANNs and the fitness function are two major issues. Both of them will be described in the following subsections.

4.5.1 Inputs and Outputs of ANNs

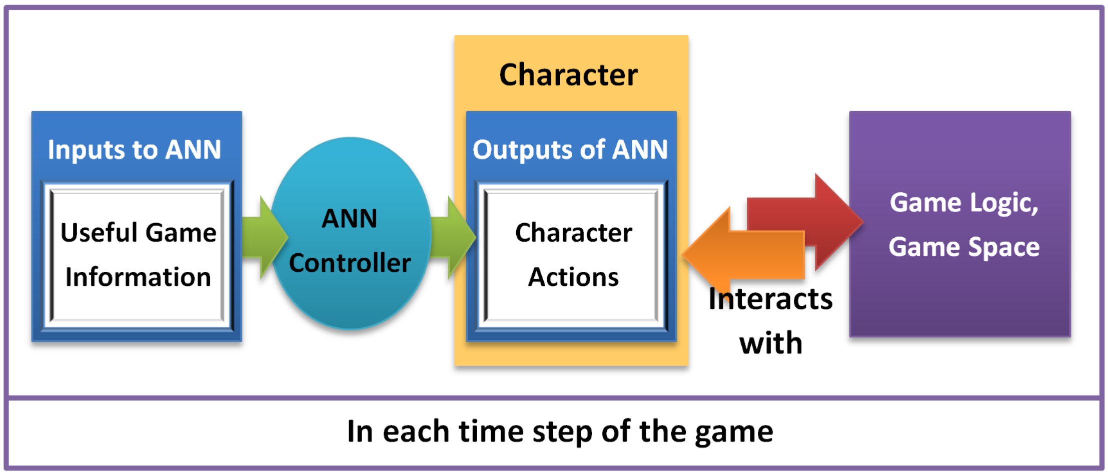


Figure 4.5: An ANN controller in a game time step.

In different games or even different characters in a game, players decide to perform actions according to the given information and their playing strategies; this makes a game or a character unique. Similarly, the information which is important for making correct decisions for a specific character should be fed as the inputs to its ANN controller. This kind of information usually includes the current status of the corresponding character, such as health points and distances to targets in a role playing game. The set of inputs should all be normalised to a unit interval of $[0, 1]$. The outputs of an ANN are usually linked to the available actions of its corresponding character. Figure 4.5 illustrates how an ANN controller is employed in a game time step. In TABS, all the ANNs of the same character have the same structure. But due to the fact that each character might need different amount of in/outputs, it is very likely that the ANN structure of one character varies from the others.

4.5.2 Fitness Function

We evaluate the performance of a player by checking his/her accomplishment in the goals and events in a game. However different games have different goals and events. Generally, we should encourage a candidate controller by increasing its fitness value if the candidate controller has attempted to achieve some goals or certain events. On the other hand, punishments as decreasing the fitness value of a training controller appropriately if the controller made a mistake could also help in developing the controllers. The candidate controllers with the highest fitness values are selected as elites in GA or used for computing the global best position in PSO. Hence, fitness function is very important that it could affect the final behaviour of a candidate controller and also the development of the whole population of the training controllers. A bad fitness function may possibly cause the controllers developing in an extremely slow pace. In the worst case, the entire population may fail to improve at all. These are the reasons why the fitness function should be carefully designed specifically for the subjected game.

Chapter 5

Case Study: MagePowerCraft

In this chapter, we demonstrate an in-house magic-combating game, MagePowerCraft, designed by us and is subjected as a case study for TABS. After introducing MagePowerCraft and TABS customization for MagePowerCraft, some validations and experiments with their results of MagePowerCraft in TABS with both GA and PSO training methods are given in the latter part of this chapter.

5.1 The Game, MagePowerCraft

The gaming environments is built up by using OGRE3D [OGR11] which is one of the most popular open-source graphics rendering engines. The 3D models are texture-mapped and rendered on the screen. The library, Hikari, is adopted for using the fancy flash GUIs in OGRE3D environment. Figure 5.2 shows the snapshots taken inside MagePowerCraft.

5.1.1 Game Rules

MagePowerCraft is a regular team-combating game much like the team player-killing (PK) systems of most modern multiplayer online action role-playing games. It supports up to a maximum of 3-on-3 team combating inside an arena. The rules of MagePowerCraft are clear:

1. The team which defeats all the opponents of the other team will score one point.

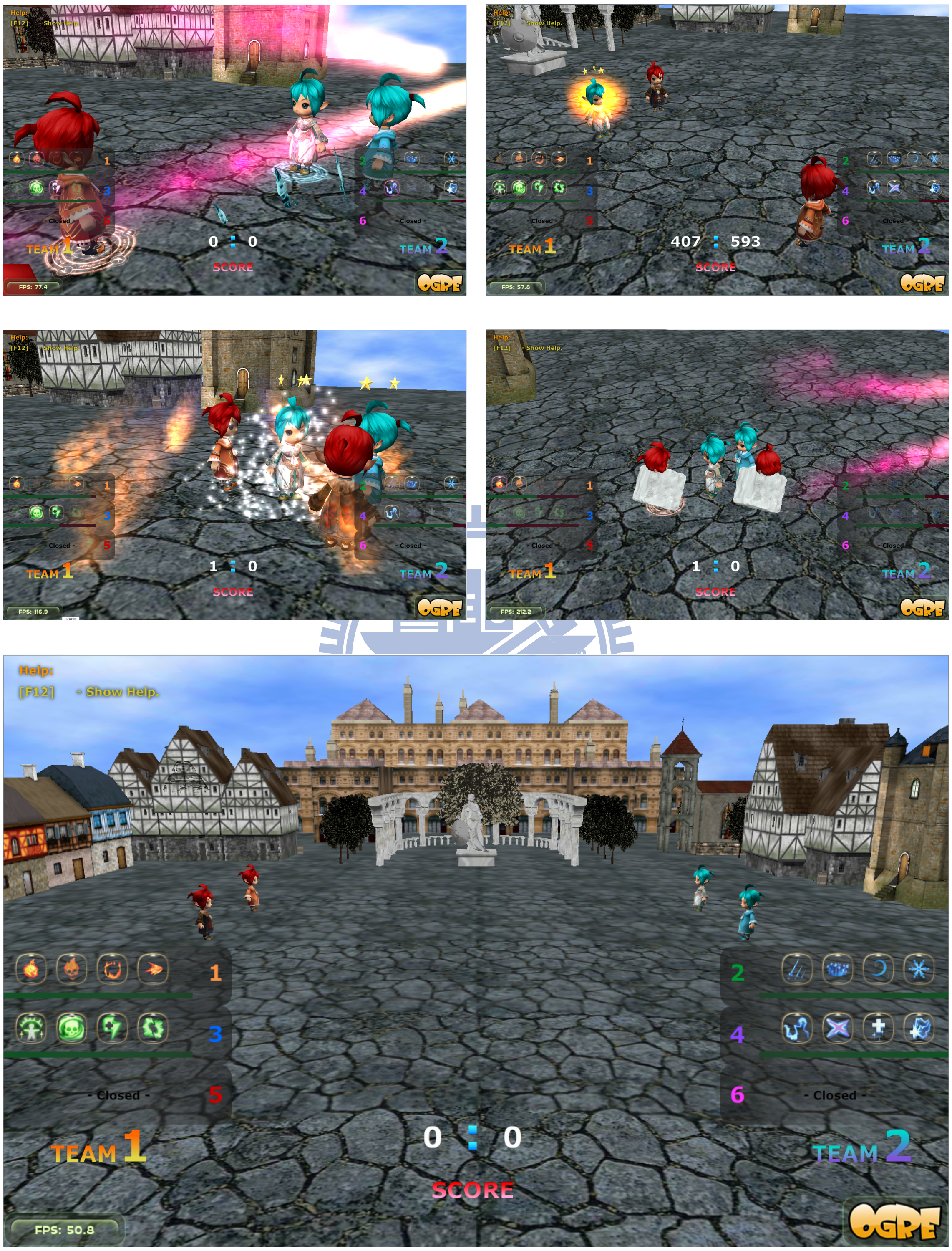


Figure 5.1: Snapshots taken from the game MagePowerCraft

2. If no team is defeated within the time limit of a match, then the match is drawn and each team is rewarded with a score of one.
3. A character is defeated once its health point (HP) reaches zero.

After a number of team battles are over, the team with the higher point would be regarded as more powerful than the other team.

5.1.2 Game Character Classes and Skills

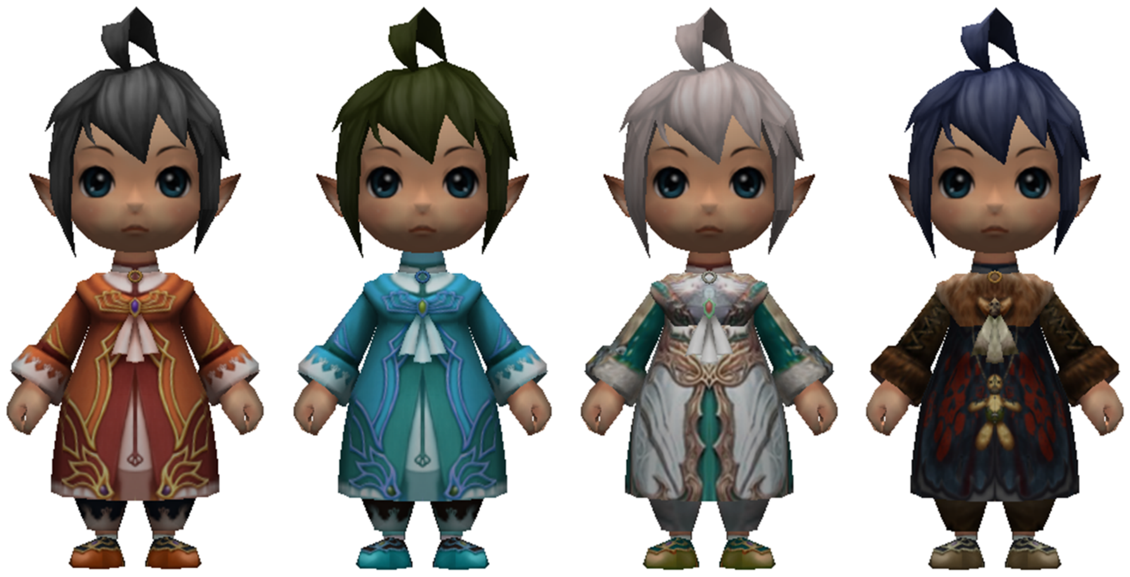


Figure 5.2: The four character classes of MagePowerCraft. From the left to the right are the Fire Wizard, the Ice Wizard, the Priest and the Shaman.

There are four classes of characters in MagePowerCraft, they are the Fire Wizard, the Ice Wizard, the Priest and the Shaman as shown in Figure 5.2. While combating, the hair colour of the characters will be changed to their team colour, red for team 1 and blue for team 2 as demonstrated in Figure 5.3. In this manner, the observers may differentiate both teams more easily. Each character class has four different skills. Although this is an in-house created game, the skills of MagePowerCraft are all rich designed. The settings of a skill may be attributed as an aiding or an attacking skill to single or multiple targets. A skill can be cast on either friendly or hostile targets.

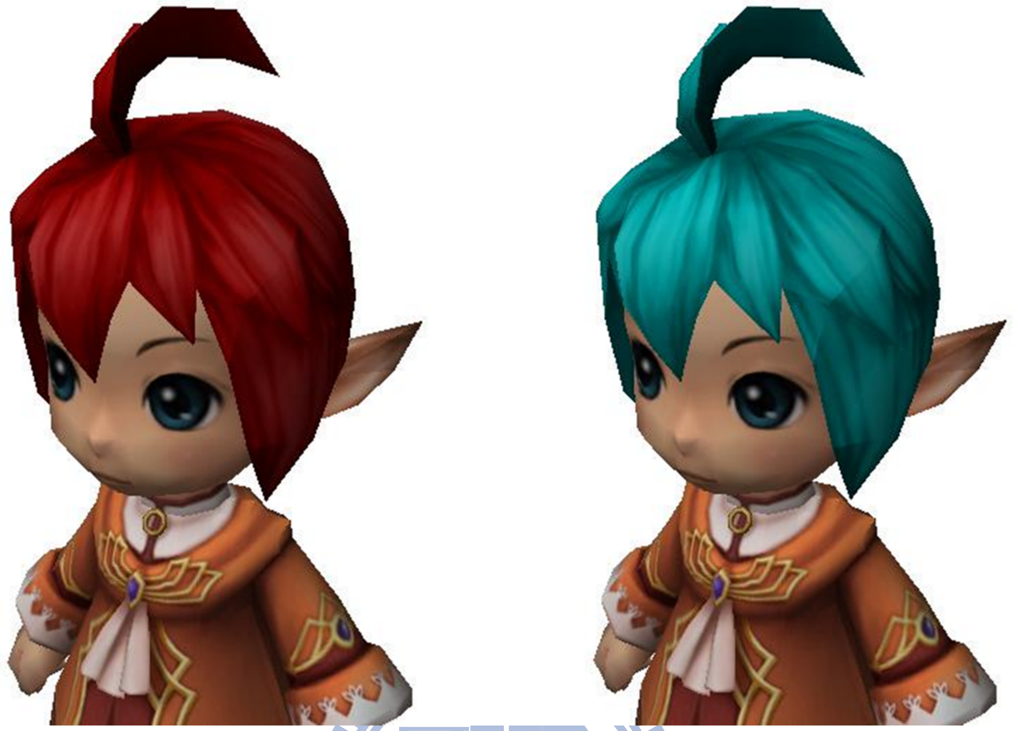


Figure 5.3: The hair of the characters will change to their team colour. Red for team 1 and blue for team 2.

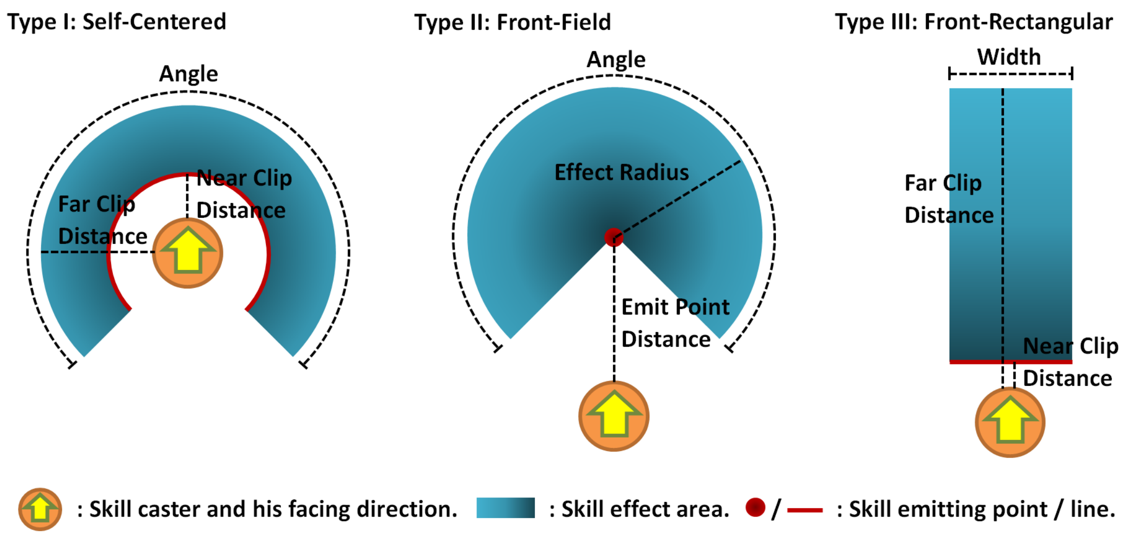


Figure 5.4: The three area of skill effect types.

There are attributes of the skills which include the power setting of the skill, area of effect (AOE) types, maximum hit, casting time, cost of mana power (MP) per cast, the cooldown (CD) time and also the special effects. The power setting of a skill includes parameters as $SkillPower$ and $SkillExtraPt$ which affect the power of the skill. The three AOE types in MagePowerCraft as shown in Figure 5.4. AOE Type I is Self-Centered, the skills of this type will effect on the area centered by the caster. Type II is Front-Field, where this kind of skills will make influence to the circular/fan area in front of the caster. And Type III is Front-Rectangular that the effecting area is a rectangular field in front of the caster. Each of the AOE type has its own parameters for creating more variations of the effects. The maximum hit of a skill indicates the maximum number of hit that a skill might cause to its target per cast. Casting time determines the time needed for casting a skill before it takes effect. Cost of MP is the amount of MP needed for each cast of the skill. CD time defines the period of wait time before a skill can be used after a prior cast of the skill. The special effects of a skill may bring different buffing (positive) or de-buffing (negative) status to the targets as shown in Table 5.1. Please check Appendix A for the detail settings of all skills of the four character classes.

Other than the various settings of the skills, all other attributes such as total HP, total MP, magical attack power and etc. of the four character classes are the same. The settings of the attributes of the four classes are listed in Table 5.2. In this way, we can focus on investigating the method for evaluating the fairness of the power of skills in MagePowerCraft using TABS.

5.1.3 Attack and Defense Formulae

A skill may cause several aiding/damage hits to its targets. The magical power of a character for each hit is computed by:

$$MagicPower = AttackPtmin + r_3(AttackPt_{max} - AttackPt_{min}), \qquad (5.1)$$

where $AttackPt_{min}$ and $AttackPt_{max}$ are the minimum and maximum of the magical

Skill Special Effect	Description
Powering Up	Enhances the Magical Attack Power of its target for a while.
Regenerating	Keeps on increasing the HP of its target periodically over a certain duration.
Burning	Keeps on deducing the HP of its target periodically over a certain duration.
Frostbiting	Slows down the movement speed of its target.
Electrocuting	Stops any action of its target intermittently over a certain duration.
Freezing	Makes its target temporarily immobilized for a while and interrupt the spells being cast by the target.
Stunning	Makes its target temporarily immobilized for a while and interrupt the spells being cast by the target.
Stunning	Makes its target temporarily immobilized for a while and interrupt the spells being cast by the target.
Poisoning	Keeps on deducing the HP of its target periodically over a certain duration.

Table 5.1: Skill Special Effects and their descriptions.

power of the caster, and r_3 is a random value inside [0, 1]. The amount of damage per hit, $DamagePt$, is given by:

$$DamagePt = max(0, MagicPower \times SkillPower + SkillExtraPt - DefensePt), \tag{5.2}$$

where $SkillPower$ and $SkillExtraPt$ are the Skill Power and Skill Extra Point, which are the parameters of the skill, and $DefensePt$ is the current defense power of the target. The total damage of the skill is $DamagePt \times HitNum$, where $HitNum$ is the actual hit number taking effect to the target. $HitNum$ is affected by $TargetToEmit$, which is the distance between the target and the skill emitting point/line. The computation

HP	MP	Basic MP	Magical Attack Power (Min)	Magical Attack Power (Max)	Magical Defense	Mana Recovery Rate	Moving Speed
20143	20836	6840	1642	1715	2875	0.585%	2.3529 m/sec

Table 5.2: Character attributes of all classes in MagePowerCraft.

of $HitNum$ is given in Equation 5.3 and 5.4, where $MaxHit$ is the maximun hit number of the skill, $Radius$ is the radius of area of effect type II, $FarClipDist$ and $NearClipDist$ are the far and near clip distance of the skill.

$$HitNum = \lfloor \frac{MaxHit \times (SkillRange - TargetToEmit)}{SkillRange} \rfloor + 1 \tag{5.3}$$

$$SkillRange = \begin{cases} Radius & \text{if the skill is area of effect type II,} \\ FarClipDist - NearClipDist & \text{otherwise.} \end{cases} \tag{5.4}$$

For the healing skills, such as the *Heal* of the Priest and the *Life Charge* of the Shaman, the $SkillPower$ is directly applied to the maximum HP of its target, $MaxHP$. Thus, the healing points per hit, $HealPt$ is given as:

$$HealPt = MaxHP \times SkillPower + SkillExtraPt \tag{5.5}$$

5.2 TABS Customization for MagePowerCraft

As we have mentioned in Section 4.5 earlier, some game-dependent parts of TABS need to be customised for each game to help developing the candidate controllers more efficiently. We describe them as follows.

5.2.1 Inputs and Outputs of ANNs in MagePowerCraft

Inputs of ANNs:

We collect the useful gaming information in MagePowerCraft for combating and feed the information as inputs to the ANNs in every simulation time step. Note that all input values are normalised to the interval of [0, 1]. We give the descriptions of the inputs along with their normalisation formulae as below:

- $\boldsymbol{Ready_i^{Norm}}$: The ready-rate of a skill i, normalised as Equation 5.6, where $TotalCD_i$ is the total CD time of skill i and $RemainCD_i$ is the remain CD time of skill i, $i = 1, 2, ..., n$, and n is the total skill number of the character.

$$Ready_i^{Norm} = (TotalCD_i - RemainCD_i)/TotalCD_i \tag{5.6}$$

- $\boldsymbol{HP_i^{Norm}}$: The current HP of all characters, including self, friendly and hostile targets, normalised as Equation 5.7, where HP_i and $MaxHP_i$ is the current and maximum HP of character i, $i = 1, 2, ..., TotalCharacterNumber$.

$$HP_i^{Norm} = HP_i/MaxHP_i \tag{5.7}$$

- $\boldsymbol{DeBuffTime_{i,j}^{Norm}}$: The remaining time of de-buffing status i of hostile target j, normalised as Equation 5.8, where $DebuffTime_{i,j}$ is the remaining time of de-buffing status i of hostile target j, $MaxDebuffTime$ is the maximum duration time of de-buffing status i, $i = 1, 2, ..., TotalDebuffStatusNumber$ and $j = 1, 2, ..., TotalHostileCharacterNumber$.

$$DebuffTime_{i,j}^{Norm} = DebuffTime_{i,j}/MaxDebuffTime_i \tag{5.8}$$

- $\boldsymbol{Dist_i^{Norm}}$: The distance between the character all other character, normalised as Equation 5.9, where $Dist_i$ is the distance between the character and target i, $i = 1, 2, ..., (TotalCharacterNumber - 1)$ and $MaxDist$ is the maximum distant of any two character, i.e. the size of the arena.

$$Dist_i^{Norm} = Dist_i/MaxDist \tag{5.9}$$

- $\boldsymbol{Casting_i^{Norm}}$: Indicating the skill that the hostile target i is casting, computed as Equation 5.10, where $SkillID$ is the mapped skill ID from 1 to $TotalSkillNum_i$

of which hostile target i is currently casting. If the hostile target i is not casting any skill, then $SkillID$ is set assigned as 0. $TotalSkillNum_i$ is the total number of skill of hostile target i, $i = 1, 2, ..., TotalHostileCharacterNumber$.

$$Casting_i^{Norm} = CastingSkillID/TotalSkillNum_i \tag{5.10}$$

Outputs of ANNs:

The output interval [0, 1] of each output is evenly divided into several sub-intervals as needed, and each sub-interval is mapped to an action or decision of the character. All the ANNs in MagePowerCraft have four outputs, they are:

- $\boldsymbol{FriendlyTgt}$: Indicating one of the friendly (including self-targeting) character target. If an aiding skill is chosen to cast by the character, then the aiding skill would be cast on this target or to the direction of this target.
- $\boldsymbol{HostileTgt}$: Indicating one of the hostile character target. If an attacking skill is chosen to cast by the character, then the attacking skill would be cast on this target or to the direction of this target.
- $\boldsymbol{MovementTgt}$: May be indicating any character target except self-targeting. This target is used as a reference target for movements of the character.
- $\boldsymbol{CastSkill}$: Indicating whether to cast a skill, if yes, which skill of the character is selected for casting.
- $\boldsymbol{Movement}$: Indicating to move forward to the $MovementTgt$, to move away from the $MovementTgt$ or to stay and not moving.

We differentiate the two cases about skill casting into friendly ($FriendlyTgt$) or hostile ($HostileTgt$) targets. An advantage is that the controllers do not need to learn about the skills that are cast on friendly targets and hostile targets, respectively. This is reasonable as an experienced player should know well about that.

5.2.2 Fitness Function for MagePowerCraft

We design a fitness function for evaluating the performance scores of the controllers for MagePowerCraft. The total fitness is computed as follows:

$$TotalFitness = max(0, Fitness_{gain} - Fitness_{loss}), \tag{5.11}$$

where $Fitness_{gain}$ and $Fitness_{loss}$ are the gain and loss values, respectively. $Fitness_{gain}$ is given by:

$$\begin{aligned} Fitness_{gain} = & AtkPt + AidPt + SkillCasting + SkillFiring + \\ & SkillBreaking + SkillEvading + LearnedSkills, \end{aligned} \tag{5.12}$$

where the variables are expained as below:

- $\boldsymbol{AtkPt}$: The damage points inflicted to the hostile targets.
- $\boldsymbol{AidPt}$: The healing points done to/from the friendly targets.
- $\boldsymbol{SkillCasting}$: The reward value for starting to cast skills. Each time when the character starts to cast a skill, a value is added to $SkillCasting$ as reward.
- $\boldsymbol{SkillFiring}$: The reward value for casting skills successfully. whenever a skill is cast and fired successfully, a value is added to $SkillFiring$ as reward.
- $\boldsymbol{SkillBreaking}$: The reward value for interrupting the skills being cast by hostile targets.
- $\boldsymbol{SkillEvading}$: The reward value for moving away from the attack range of the skills of the hostile targets.
- $\boldsymbol{LearnedSkills}$: The reward value for learning to cast different skills.

The fitness value is decreased by $Fitness_{loss}$ which is given as:

$$Fitness_{loss} = HurtPt + CastWhileCD + OutOfRange + SkillBroken, \tag{5.13}$$

where the variables are explained as follows:

- $\boldsymbol{HurtPt}$: The damage points inflicted by the hostile targets.
- $\boldsymbol{CastWhileCD}$: The punishment for casting skills in while the skills are still in CD.
- $\boldsymbol{OutOfRange}$: The punishment for casting skills on the invalid targets which are too far away from the character and out of range the range of the casting skills.
- $\boldsymbol{SkillBroken}$: The punishment for being interrupted by the hostile targets while casting skills.

5.3 Validations, Experiments and Results

A game might need some validations to ensure that TABS is feasible to help balancing for a particular game, therefore, we performed three validations to test the feasibility of balancing MagePowerCraft with TABS. A system error, namely ϵ_{TABS}, can be acquired after the validations. If the error value is acceptable, then we may claim that TABS is feasible for helping to balance the game. After the validations, we performed three experiments as examples of balancing the teams with different demands of character class combinations.

We trained a population of 100 candidate controllers with both GA and PSO for each character in each validation and experiment. After the training sessions, we applied the best-trained controllers to control the characters to play the games. To increase the reliability of the statistics, we examined the behaviours of the controllers and collected 10 sets of the valid results after the convergence of a training for each validation and experiment. For each set, 1000 games were played by the controllers and the statistics of the combating result were recorded. The average balancing error, ϵ_{avg}, is computed as Equation 5.14, where $AvgScore$ is the average score of the 10 sets and $NumGames$ = 1000.

$$\epsilon_{avg} = |AvgScore - NumGames/2|/(NumGames/2) \tag{5.14}$$

All validations and experiments were performed on a desktop PC with Quad Core CPU - Intel (R) Core (TM) i7 CPU 870 @ 2.93GHz and the OS was Windows 7 Professional.

5.3.1 TABS Validations

We validated TABS in three cases with different settings. For each case, We assigned the same character set to the two teams. Obviously, the ability power of the both teams are balanced. Thus, the balanced conclusion should be drawn by TABS, i.e., the ϵ_{avg} value of each case should all be small. The two teams in the three cases are:

1. (Team 1: Fire Wizard) verses (Team 2: Fire Wizard)
2. (Team 1: Fire Wizard + Ice Wizard) verses (Team 2: Fire Wizard + Ice Wizard)
3. (Team 1: Fire Wizard + Ice Wizard + Priest) verses (Team 2: Fire Wizard + Ice Wizard + Priest)

Training Model: GA

The following tables show the validation results of the three cases by applying GA:

Validation: Case 1

Game Set	1	2	3	4	5
Team1:Team2	491:509	490:510	483:517	474:526	527:473
Game Set	6	7	8	9	10
Team1:Team2	510:490	497:503	499:501	492:508	491:509
Average	495.4:504.6		Error ϵ_{avg}	0.92%	

Table 5.3: The results of Validation by GA, Case 1.

Validation: Case 2

Game Set	1	2	3	4	5
Team1:Team2	518:482	488:512	497:503	523:477	507:493
Game Set	6	7	8	9	10
Team1:Team2	482:518	484:516	490:510	527:473	477:523
Average	499.3:500.7		Error ϵ_{avg}	0.14%	

Table 5.4: The results of Validation by GA, Case 2.

Validation: Case 3

Game Set	1	2	3	4	5
Team1:Team2	515:485	520:480	521:479	487:513	487:513
Game Set	6	7	8	9	10
Team1:Team2	490:510	496:504	459:541	480:520	488:512
Average	494.3:505.7		Error ϵ_{avg}	1.14%	

Table 5.5: The results of Validation by GA, Case 3.

Training Model: PSO

The tables below give the validation results of the three cases by applying PSO:

Validation: Case 1

Game Set	1	2	3	4	5
Team1:Team2	495:505	507:493	499:501	498:502	498:502
Game Set	6	7	8	9	10
Team1:Team2	490:510	515:485	521:479	480:520	516:484
Average	501.9:498.1		Error ϵ_{avg}	0.38%	

Table 5.6: The results of Validation by PSO, Case 1.

Validation: Case 2

Game Set	1	2	3	4	5
Team1:Team2	508:492	508:492	498:502	518:482	504:496
Game Set	6	7	8	9	10
Team1:Team2	508:492	484:516	492:508	468:532	507:493
Average	499.5:500.7		Error ϵ_{avg}	0.1%	

Table 5.7: The results of Validation by PSO, Case 2.

Validation: Case 3

Game Set	1	2	3	4	5
Team1:Team2	507:493	515:485	510:490	489:511	502:498
Game Set	6	7	8	9	10
Team1:Team2	516:484	496:504	510:490	508:492	489:511
Average	504.2:495.8		Error ϵ_{avg}	0.84%	

Table 5.8: The results of Validation by PSO, Case 3.

Consider that GA was used, the average balance error values, ϵ_{avg}, of the three cases are 0.92%, 0.14% and 1.14%, respectively. Thus, the system error value by using GA is the largest value among them, i.e. $\epsilon^{GA}_{TABS} = 1.14\%$. And for PSO, the average balance error values of the three cases are 0.38%, 0.1% and 0.84%, respectively. Hence, the system error value by using PSO $\epsilon^{PSO}_{TABS} = 0.84\%$.

The results show that the validity of TABS to MagePowerCraft by both training models is satisfactory. The two system errors, ϵ^{GA}_{TABS} and ϵ^{PSO}_{TABS}, are for determining whether or not the ability power of two teams are similar in MagePowerCraft by using the corresponding training model. That is to say, if the error of a testing result is lower than the corresponding system error of the training model, we conclude that the ability power of the two teams is balanced. Otherwise, the two teams are judged as unbalanced. Notice that the system error values are subjected to change for different games with different features.

5.3.2 Experiments

We take the role as game designers and demonstrate three independent experiments of using TABS to tune the skill settings of the character classes in MagePowerCraft (See Appendix A). The goal of our experiments are to adjust the skill parameters and make the skill ability power of one team matching with the other in each case, i.e. the final ϵ_{avg} of each case should be smaller than the corresponding system error, ϵ^{GA}_{TABS} and ϵ^{PSO}_{TABS}, which we had obtained earlier from the validations in Section 5.3.1. Each experiment is done by GA first and then repeated again by using PSO The team combanition of the three experiments are:

1. (Team 1: Fire Wizard) verses (Team 2: Ice Wizard)
2. (Team 1: Fire Wizard + Shaman) verses (Team 2: Ice Wizard + Priest)
3. (Team 1: Fire Wizard) verses (Team 2: Ice Wizard + Priest + Shaman)

Experiment 1:

In this experiment, we would like to show the character-to-character level balancing. Before adjusting the skill parameters, we need to evaluate the balance situation of the two characters, and below are the results with the original skill settings by using GA and PSO

Game Set	1	2	3	4	5
Team1:Team2	615:385	620:380	648:352	586:414	597:403
Game Set	6	7	8	9	10
Team1:Team2	627:373	632:368	623:377	583:417	633:367
Average	616.4:383.6		Error ϵ_{avg}	23.28%	

Table 5.9: The results of Experiment 1, with original skill settings by applying GA

Game Set	1	2	3	4	5
Team1:Team2	594:406	613:387	627:373	635:365	656:344
Game Set	6	7	8	9	10
Team1:Team2	673:327	657:343	602:398	606:394	648:352
Average	631.1:368.9		Error ϵ_{avg}	26.22%	

Table 5.10: The results of Experiment 1, with original skill settings by applying PSO

Apparently, the two teams were not balanced (23.28% > ϵ_{TABS}^{GA} = 1.14% and 26.22% > ϵ_{TABS}^{PSO} = 0.84%). Based on the scores, we find out that Team 1 was more powerful than Team 2. To balance the ability power of two teams, as an example, we choose to strengthen the power of *Freezing Sword* and *Blizzard* of the Ice Wizard in Team 2, while we weaken the *Fire Ball* skill of Fire Wizard in Team 1 at the same time. It took several steps for us to fine-tune the skill parameters. Once a new setting is made, we run it with TABS and see if the new setting is balanced. And after a few tries, we eventually came up with the setting the (*Skill Power*,*Extra Skill Point*) of the skill *Freezing Sword*, *Blizzard* and *Fire Ball* as (244%, 556), (540%, 574) and (322%, 360), respectively. This new setting gives the balanced results as listed in Table 5.11 and 5.12 for G.A and PSO

Game Set	1	2	3	4	5
Team1:Team2	500:500	512:488	511:489	534:466	480:520
Game Set	6	7	8	9	10
Team1:Team2	505:495	501:499	483:517	516:484	487:513
Average	502.9:497.1		Error ϵ_{avg}	0.58%	

Table 5.11: The results of Experiment 1, with new skill settings by applying GA

Game Set	1	2	3	4	5
Team1:Team2	501:499	493:507	514:486	518:482	491:509
Game Set	6	7	8	9	10
Team1:Team2	500:500	500:500	508:492	512:488	501:499
Average	503.8:496.2		Error ϵ_{avg}	0.76%	

Table 5.12: The results of Experiment 1, with new skill settings by applying PSO

Experiment 2:

In this experiment, we would like to show the team-to-team level balancing. We evaluated the balance situation of the two teams by TABS with the original settings, and down below are the results.

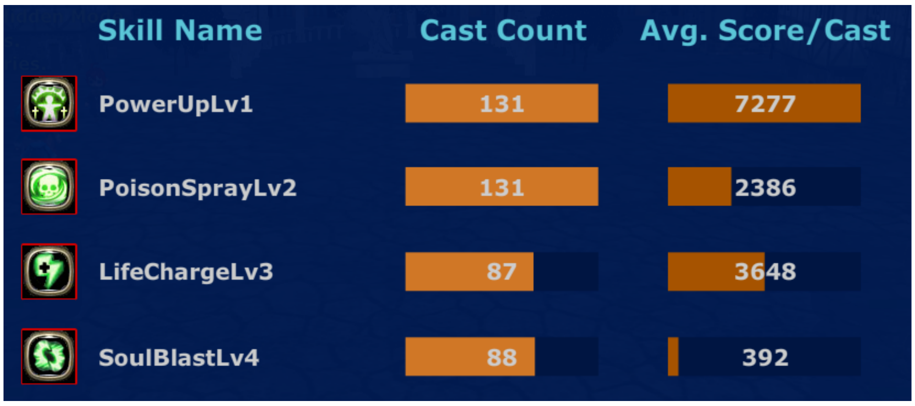
Game Set	1	2	3	4	5
Team1:Team2	515:485	491:509	429:571	497:503	491:509
Game Set	6	7	8	9	10
Team1:Team2	471:529	475:525	499:501	490:510	474:526
Average	483.2:516.8		Error ϵ_{avg}	3.36%	

Table 5.13: The results of Experiment 2, with original skill settings by applying GA

Game Set	1	2	3	4	5
Team1:Team2	461:539	452:548	516:484	455:545	458:542
Game Set	6	7	8	9	10
Team1:Team2	458:542	459:541	462:538	497:503	467:533
Average	469.4:530.6		Error ϵ_{avg}	6.12%	

Table 5.14: The results of Experiment 2, with original skill settings by applying PSO

The error shows that the two teams were not balanced (3.36% > ϵ^{GA}_{TABS} = 1.14% and 6.12% > ϵ^{PSO}_{TABS} = 0.84%). After checking the gaming statistics of Shaman recorded by TABS as shown in Figure 5.5, we noticed that the average score per cast of skill *Soul Blast*



Skill Name	Cast Count	Avg. Score/Cast
PowerUpLv1	131	7277
PoisonSprayLv2	131	2386
LifeChargeLv3	87	3648
SoulBlastLv4	88	392

Figure 5.5: The gaming statistics of Shaman with the origin skill setting.

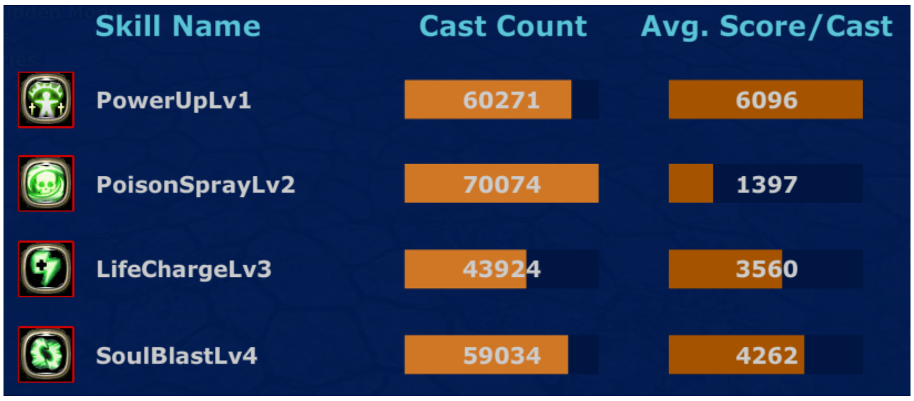
by Shaman in Team 1 is relatively low. Since a low average score per cast could indicate that the power of a skill may be too low. Therefore, we decided to enhance the attack power of *Soul Blast* in this experiment. This time we quickly found the balanced setting for only one attempt of trying, which is to set the (*Skill Power*,*Extra Skill Point*, *Max Hit*) of (*Soul Blast*) from (171%, 274, 5) to (385%, 314, 6). The results with the new settings are listed below. Figure 5.6 shows the gaming statistics of Shaman after the new settings.

Game Set	1	2	3	4	5
Team1:Team2	477:523	472:528	494:506	499:501	507:493
Game Set	6	7	8	9	10
Team1:Team2	487:513	526:474	516:484	494:506	492:508
Average	496.4:503.6		Error ϵ_{avg}	0.72%	

Table 5.15: The results of Experiment 2, with new skill settings by applying GA

Game Set	1	2	3	4	5
Team1:Team2	469:531	492:508	526:474	488:512	508:492
Game Set	6	7	8	9	10
Team1:Team2	519:481	503:497	524:476	467:533	498:502
Average	499.4:500.6		Error ϵ_{avg}	0.12%	

Table 5.16: The results of Experiment 2, with new skill settings by applying PSO



Skill Name	Cast Count	Avg. Score/Cast
PowerUpLv1	60271	6096
PoisonSprayLv2	70074	1397
LifeChargeLv3	43924	3560
SoulBlastLv4	59034	4262

Figure 5.6: The gaming statistics of Shaman with the new skill setting.

Experiment 3:

In the last experiment, we would like to show the character-team balancing. We may assume that this experiment is the situation of trying to balance between a single non-player-controlled character (NPC) boss, the Fire Wizard, and a team of player characters. As usual, we evaluate the balance situation of the two sides first. The results of the evaluations are as follows.

Game Set	1	2	3	4	5
Team1:Team2	515:485	491:509	429:571	497:503	491:509
Game Set	6	7	8	9	10
Team1:Team2	471:529	475:525	499:501	490:510	474:526
Average	483.2:516.8		Error ϵ_{avg}	3.36%	

Table 5.17: The results of Experiment 3, with original skill settings by applying GA

Game Set	1	2	3	4	5
Team1:Team2	461:539	452:548	516:484	455:545	458:542
Game Set	6	7	8	9	10
Team1:Team2	458:542	459:541	462:538	497:503	467:533
Average	469.4:530.6		Error ϵ_{avg}	6.12%	

Table 5.18: The results of Experiment 3, with original skill settings by applying PSO

Team 1 with only one character is obviously weaker than Team 2 which has three characters. Because that most NPC boss has high HP value than the player characters, thus, this time we will adjust the maximum HP of the Fire Wizard and letting it to match the power ability of the team of three players. After several adjustments, the two teams are finally balanced as the result listed in Table 5.19 and 5.20 with setting the maximum HP of the Fire Wizard as 55393.

Game Set	1	2	3	4	5
Team1:Team2	518:482	490:510	505:495	525:475	485:515
Game Set	6	7	8	9	10
Team1:Team2	503:497	487:513	513:487	511:489	492:508
Average	502.9:497.1		Error ϵ_{avg}	0.58%	

Table 5.19: The results of Experiment 3, with new skill settings by applying GA

Game Set	1	2	3	4	5
Team1:Team2	513:487	506:494	491:509	504:496	492:508
Game Set	6	7	8	9	10
Team1:Team2	502:498	518:482	511:489	506:494	490:510
Average	503.3:496.7		Error ϵ_{avg}	0.66%	

Table 5.20: The results of Experiment 3, with new skill settings by applying PSO

Chapter 6

Discussion

6.1 Comparing with Other Game Balancing Methods

Comparing to the time and human resource consuming process of traditional game tuning by test series, our approach only took at most three hours for the whole process in each experiment including all the adjustments (See Section 5.3.2). Moreover, other than the scripted AI which suffers from the bias due to the routine-like actions, TABS can train controllers with different behaviours that suit for fighting against their opponents or co-operating with their teammates in an un-predefined manner.

6.2 GA Versus PSO in TABS

Recently, particle swarm optimization is proved to be better than GA in some aspects (Hassan et al. [HCDWV05]). Some researches show that PSO tends to compute the optimized results faster and has better computation efficiency than other evolutionary optimizations, such as GA. But in TABS, the training time difference between GA and PSO is small, since that updating GA or PSO only takes a small portion of the total training time, where the training time is dominated by the process of game simulations. Table 6.1 gives the average training time per game and along with the ANN configuration of different combat types.

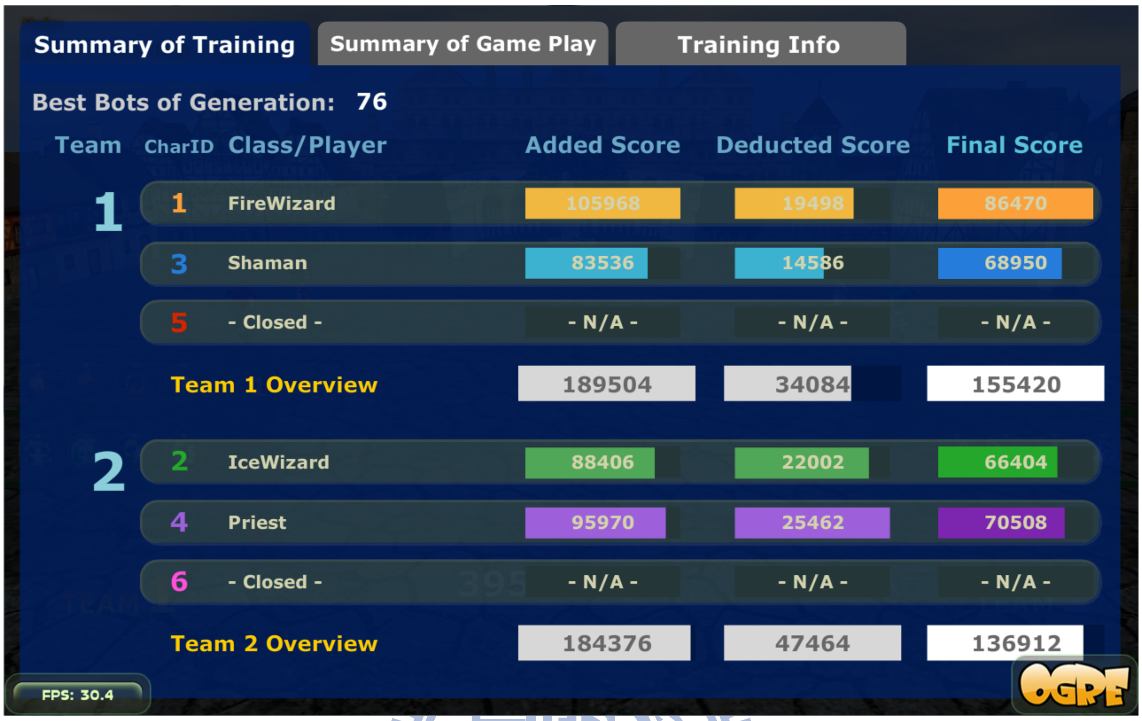
Combat	Average Training Time per Game		ANN Configuration		
Types	GA	PSO	#Input	#Oututs	#Weights
1-on-1	1.2 sec	1.2 sec	10	5	90
2-on-2	6.5 sec	6.4 sec	17	5	132
3-on-3	18.4 sec	19.1 sec	24	5	174

Table 6.1: Average training time per game and ANN configuration of different combat types.

The behaviour abilities of the controllers trained by either GA or PSO have no significant difference. Both training models usually gets converged results before 150 generations/iterations. The result of evaluation and balancing with both training models are also both satisfactory. Thus, we cannot tell which one is more better than the other one.

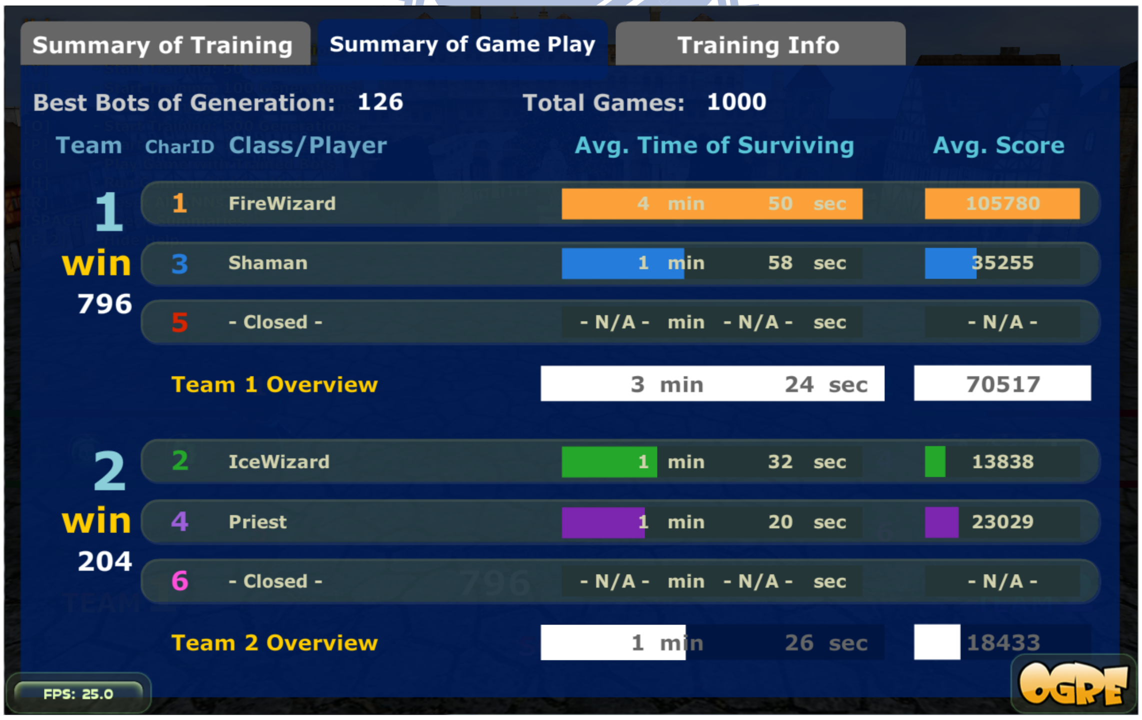
6.3 Intra-team Balancing

With the help of TABS, we may evaluate and compare the performance of the two teams by computing performance scores if a member of a team performs certain actions. The criteria for evaluating the performance scores of a game can also be extended to other aspects. For example, we may consider intra-team balancing. Imagine the case that a character is easily killed at the very beginning of the combats. We believe that this is not a good gaming experience for the player who controlled that character even if the team abilities are balanced. In this case, TABS can show the gaiming statistics information, such as the average surviving time, the average score of each character and the average score per cast of the skills as we had mentioned earlier while balancing in Experiment 2. This kind of information can be quite useful with intra-team balancing. Figure 6.1 show that the gaming information are recorded and provided by TABS.



Best Bots of Generation: 76

Team	CharID	Class/Player	Added Score	Deducted Score	Final Score
1	1	FireWizard	105968	19498	86470
	3	Shaman	83536	14586	68950
	5	- Closed -	- N/A -	- N/A -	- N/A -
		Team 1 Overview	189504	34084	155420
2	2	IceWizard	88406	22002	66404
	4	Priest	95970	25462	70508
	6	- Closed -	- N/A -	- N/A -	- N/A -
		Team 2 Overview	184376	47464	136912



Best Bots of Generation: 126 Total Games: 1000

Team	CharID	Class/Player	Avg. Time of Surviving	Avg. Score
1 win 796	1	FireWizard	4 min 50 sec	105780
	3	Shaman	1 min 58 sec	35255
	5	- Closed -	- N/A - min - N/A - sec	- N/A -
		Team 1 Overview	3 min 24 sec	70517
2 win 204	2	IceWizard	1 min 32 sec	13838
	4	Priest	1 min 20 sec	23029
	6	- Closed -	- N/A - min - N/A - sec	- N/A -
		Team 2 Overview	1 min 26 sec	18433

Figure 6.1: Gaming information recorded by TABS.

6.4 Limitations of TABS

However, there are some limitations of the Trian-by-Coaches training scheme. Although we had automatic training in TABS, human inspecting is still required for the behavior and maturity of the final best controllers. The main reason is that the best controllers are not guaranteed as the best controllers in general cases. Since the current best controllers are trained only to deal with the best controllers of the last generation/iteration (i.e. the current coach controllers), leading the best controllers might be good when only confronting with the coach controllers who trained them but not with any other controllers. In other words, sometimes the best controllers might be over-adapted to their last coach controllers. Once these over-adapted controllers combat with the best controllers from other characters, they might have very poor performance. This over-adaption of the best controllers may also be one of the main reasons that cause the best fitness growth line jittery. Figure 6.2 shows one example of the best fitness growth line of a character trained by TABS.

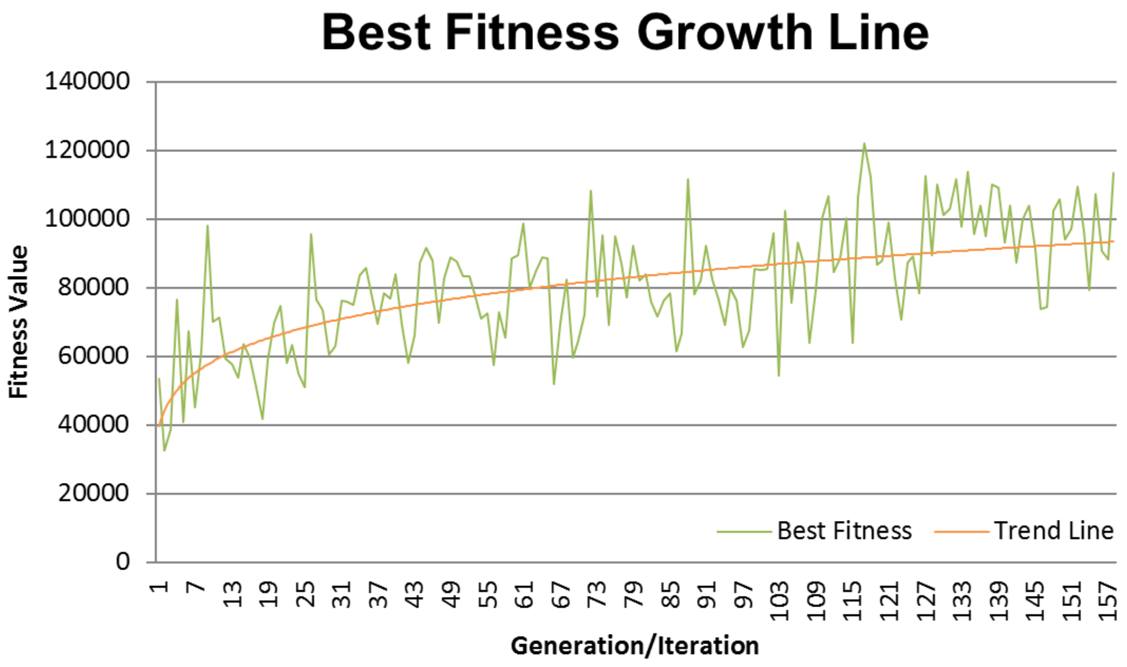


Figure 6.2: The best fitness growth line of a character trained by TABS.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

We introduce a novel Team Ability Balancing System (TABS) which evaluates the fairness of teams in a role-playing combating game. TABS evaluate the fairness of a game by training ANN controllers to combat with each other by either using genetic algorithm or particle swarm optimization. Only the best-trained controllers of each character are used for fairness evaluation. Besides the score of both teams, the statistics collected by TABS are also useful for game designers to tune the gaming parameters for both inter and intra team balancing. We have validated and performed experiments by applying it to our in-house game, MagePowerCraft. And the results of validations and experiments are satisfactory. As the controllers are evolved with Train-by-Coaches training scheme, the controllers can all be trained in a fair fashion. Furthermore, Parallel training in separate game spaces also helped to speed up the training process. We show that TABS is useful in assisting to balance the team abilities efficiently.

7.2 Future Work

It is an interesting research direction to find a better and promising solution for verifying the maturity of the controllers automatically without human inspecting. We want to develop a fully automatic evaluating and adjusting system for ability balance in team combating games in the future. Besides, we would like to develop an enhanced Train-by-Coaches training scheme in order to solve the problem of candidate controllers over-adapting to the coach controllers. We may try to increase the diversity of the coach controllers while training, such as trying to let a candidate controller not only trained by the coaches of current generation/iteration, but also the coaches of previous generations/iterations.

In addition to finding a better training scheme, we would also like to test our system in a game with larger scales, such as more player, more skills per character class and more complexity of the skills. But larger scale may indicate that more inputs or outputs for an ANN controller, then the total number of weight in an ANN (i.e. also the dimension of the solution) would be increasing dramatically. This may lead to a result of a slow convergence rate. If possible, we would like to find a feasible training model that suit for high dimensional search space for TABS.

Appendix A

Skill Settings in MagePowerCraft

A.1 Fire Wizard

Skill	Description
Fire Ball Attacking Skill	Skill Power: 501%, Extra Skill Point: 414 Cost MP: Basic MP $\times$ 2.1% Target: Single Attack Range: 0 $\sim$ 10.59m Max Damage Hit: 1 Hit Cast Time: 2 sec CD Time: 17 sec Special Effect Burning: Takes 10% of last hit damage point every 2 sec and last for 8 sec. Stunning: Last for 1.5 sec.
Inferno	Skill Power: 185%, Extra Skill Point: 276 Cost MP: Basic MP $\times$ 3.4% Target: Multiple, AOE Type III: Front-Rectangular Attack Range: 0.5 $\sim$ 3.53m (Near $\sim$ Far), Width: 0.6m

Attacking Skill	Max Damage Hit: 12 Hit Cast Time: 0.8 sec CD Time: 24 sec Special Effect Burning: Takes 50% of last hit damage point every 2 sec and last for 10 sec. Stunning: Last for 4 sec.
Fire Wall Attacking Skill	Skill Power: 184%, Extra Skill Point: 162 Cost MP: Basic MP × 3.2% Target: Multiple, AOE Type I: Self-Centered Attack Range: 0 ~ 2.59 m (Near ~ Far), Angle: 360° Max Damage Hit: 6 Hit Cast Time: 1.2 sec CD Time: 33 sec Special Effect Burning: Takes 5% of last hit damage point every 2 sec and last for 10 sec. Stunning: Last for 3 sec.
Fire Shots Attacking Skill	Skill Power: 211%, Extra Skill Point: 134 Cost MP: Basic MP × 2.5% Target: Multiple, AOE Type III: Front-Rectangular Attack Range: 0 ~ 10.59 m (Near ~ Far), Width: 0.6 m Max Damage Hit: 3 Hit Cast Time: 0.1 sec CD Time: 20 sec Special Effect Burning: Takes 10% of last hit damage point every 2 sec and last for 15 sec.

A.2 Ice Wizard

Skill	Description
Freezing Sword Attacking Skill	Skill Power: 214%, Extra Skill Point: 556 Cost MP: Basic MP $\times$ 2.3% Target: Multiple, AOE Type III: Front-Rectangular Attack Range: 0 $\sim$ 2.35 m (Near $\sim$ Far), Width: 0.6 m Max Damage Hit: 3 Hit Cast Time: 0.1 sec CD Time: 15 sec Special Effect Frostbiting: $MovingSpeed$ deduct $\times$ 50% and last for 2 sec. Stunning: Last for 3.5 sec.
Freezing Field Attacking Skill	Skill Power: 279%, Extra Skill Point: 462 Cost MP: Basic MP $\times$ 3.1% Target: Multiple, AOE Type II: Front-Field Emit Distance: 4 m, Radius: 1.41 m, Angle: 360° Max Damage Hit: 1 Hit Cast Time: 1.1 sec CD Time: 28 sec Special Effect Frostbiting: $MovingSpeed$ deduct $\times$ 50% and last for 8 sec.
Blizzard Attacking Skill	Skill Power: 210%, Extra Skill Point: 334 Cost MP: Basic MP $\times$ 2.3% Target: Multiple, AOE Type II: Front-Field Emit Distance: 1.6 m, Radius: 1 m, Angle: 360° Max Damage Hit: 2 Hit Cast Time: 0.9 sec CD Time: 15 sec

	Special Effect Frostbiting: $MovingSpeed$ deduct × 50% and last for 8 sec. Stunning: Last for 0.5 sec.
Instant Freeze Attacking Skill	Skill Power: 226%, Extra Skill Point: 270 Cost MP: Basic MP × 2.3% Target: Multiple, AOE Type I: Self-Centered Attack Range: 0 ∼ 1.88 m (Near ∼ Far), Angle: 360° Max Damage Hit: 1 Hit Cast Time: 0.5 sec CD Time: 24 sec Special Effect Freezing: Last for 2 sec.

A.3 Priest

Skill	Description
Chain Lightning Attacking Skill	Skill Power: 220%, Extra Skill Point: 85 Cost MP: Basic MP × 2.5% Target: Multiple, AOE Type I: Self-Centered Attack Range: 0 ∼ 5.89 m (Near ∼ Far), Angle: 360° Max Damage Hit: 3 Hit Cast Time: 0.8 sec CD Time: 20 sec Special Effect Electrocuting: Breaks action every 5 sec and last for 12 sec.

Grand Cross Attacking Skill	Skill Power: 169%, Extra Skill Point: 262 Cost MP: Basic MP × 3.2% Target: Multiple, AOE Type III: Front-Rectangular Attack Range: 0.5 ~ 11.29 m (Near ~ Far), Width: 1.8 m Max Damage Hit: 6 Hit Cast Time: 2.6 sec CD Time: 28 sec <u>Special Effect</u> Stunning: Last for 1 sec.
Heal Aiding Skill	Skill Power: 8%, Extra Skill Point: 324 Cost MP: Basic MP × 3% Target: Multiple, AOE Type I: Self-Centered Effect Range: 0 ~ 4 m (Near ~ Far), Angle: 360° Max Effect Hit: 1 Hit Cast Time: 0.5 sec CD Time: 60 sec
Lightning Volt Attacking Skill	Skill Power: 181%, Extra Skill Point: 294 Cost MP: Basic MP × 2.7% Target: Multiple, AOE Type III: Front-Rectangular Attack Range: 0 ~ 4.94 m (Near ~ Far), Width: 0.6 m Max Damage Hit: 5 Hit Cast Time: 0.5 sec CD Time: 18 sec <u>Special Effect</u> 60% Electrocuting: Breaks action every 5 sec and last for 13 sec.

A.4 Shaman

Skill	Description
Power Up Aiding Skill	Skill Power: 10%, Extra Skill Point: 0 Cost MP: Basic MP × 3% Target: Multiple, AOE Type I: Self-Centered Effect Range: 0 ∼ 2.5 m (Near ∼ Far), Angle: 360° Max Effect Hit: 1 Hit Cast Time: 0.5 sec CD Time: 40 sec Special Effect Powering Up: Magical Attack Power enhanced by 10% for 15 sec.
Poison Spray Attacking Skill	Skill Power: 196%, Extra Skill Point: 262 Cost MP: Basic MP × 2.7% Target: Multiple, AOE Type I: Self-Centered Attack Range: 0 ∼ 3.12 m (Near ∼ Far), Angle: 60° Max Damage Hit: 3 Hit Cast Time: 0.4 sec CD Time: 22 sec Special Effect Poisoning: Takes 20% of last hit damage point every 2.5 sec and last for 11 sec.
Life Charge Aiding Skill	Skill Power: 3.5%, Extra Skill Point: 436 Cost MP: Basic MP × 4% Target: Multiple, AOE Type I: Self-Centered Effect Range: 0 ∼ 4.2 m (Near ∼ Far), Angle: 360° Max Effect Hit: 1 Hit Cast Time: 1 sec CD Time: 60 sec

	Special Effect Regenerated: Regenerate 3.5% of Maximum HP every 2 sec and last for 9 sec.
Soul Blast Attacking Skill	Skill Power: 171%, Extra Skill Point: 274 Cost MP: Basic MP $\times$ 2.9% Target: Multiple, AOE Type I: Self-Centered Attack Range: 0.5 $\sim$ 5.02 m (Near $\sim$ Far), Angle: 360° Max Damage Hit: 5 Hit Cast Time: 1.5 sec CD Time: 21 sec

Bibliography

[Ang98] P. Angeline. Evolutionary optimization versus particle swarm optimization: Philosophy and performance differences. In *Evolutionary Programming VII*, pages 601--610, 1998.

[Bra95] J. Branke. Evolutionary algorithms for neural network design and training. In *Nordic Workshop on Genetic Algorithms and its Applications*, 1995.

[CLL09] L. Cardamone, D. Loiacono, and P.L. Lanzi. Evolving competitive car controllers for racing games with neuroevolution. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 1179--1186, 2009.

[CLM04] N. Cole, S. Louis, and C. Miles. Using a genetic algorithm to tune first-person shooter bots. In *Proceedings of the International Congress on Evolutionary Computation*, volume 1, pages 139--145, 2004.

[CP98] E. Cantú-Paz. A survey of parallel genetic algorithms. *Calculateurs Paralleles, Reseaux et Systems Repartis*, 10(2):141--171, 1998.

[ES98] R. Eberhart and Y. Shi. Comparison between genetic algorithms and particle swarm optimization. In *Evolutionary Programming VII*, pages 611--616, 1998.

[ES01] R.C. Eberhart and Y. Shi. Particle swarm optimization: developments, applications and resources. In *Evolutionary computation*, volume 1, pages 81--86, 2001.

[FMD02] K.D. Forbus, J.V. Mahoney, and K. Dill. How qualitative spatial reasoning can improve strategy game AIs. *IEEE Intelligent Systems*, 17(4):25--30, 2002.

[FP10] J. Fürnkranz and A. Pfeifer. Creating adaptive game ai in a real time continuous environment using neural networks. 2010.

[G$^+$89] D.E. Goldberg et al. *Genetic algorithms in search, optimization, and machine learning*. 1989.

[GMS03] R. Graham, H. McCabe, and S. Sheridan. Pathfinding in computer games. *ITB Journal*, 8, 2003.

[HC04] R. Hunicke and V. Chapman. AI for dynamic difficulty adjustment in games. In *Challenges in Game Artificial Intelligence AAAI Workshop*, pages 91--96, 2004.

[HCDWV05] R. Hassan, B. Cohanim, O. De Weck, and G. Venter. A comparison of particle swarm optimization and the genetic algorithm. In *Proceedings of the 1st AIAA Multidisciplinary Design Optimization Specialist Conference*, 2005.

[Hol92] J.H. Holland. Genetic algorithms. *Scientific American*, 267(1):66--72, 1992.

[KE95] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Neural Networks*, volume 4, pages 1942--1948, 1995.

[Ken97] J. Kennedy. The particle swarm: social adaptation of knowledge. In *Evolutionary Computation*, pages 303--308, 1997.

[LSL08] R. Leigh, J. Schonfeld, and S.J. Louis. Using coevolution to understand and validate game balance in continuous games. In *Genetic and evolutionary computation*, pages 1563--1570, 2008.

[MN98] M. Matsumoto and T. Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. on Modeling and Computer Simulation*, 8(1):3--30, 1998.

[OGR11] OGRE3D. Ogre, October 2011. `http://www.ogre3d.org/`.

[OYH08] J.K. Olesen, G.N. Yannakakis, and J. Hallam. Real-time challenge balance in an rts game using rtneat. In *Computational Intelligence and Games*, pages 87--94, 2008.

[Rie01] T. Riechmann. Genetic algorithm learning and evolutionary games. *Journal of Economic Dynamics and Control*, 25(6-7):1019--1037, 2001.

[RM02] T.E. Revello and R. McCartney. Generating war game strategies using a genetic algorithm. In *Proceedings of the 2002 Congress on Evolutionary Computation*, volume 2, pages 1086--1091, 2002.

[RN04] M. Rocha and J. Neves. Preventing premature convergence to local optima in genetic algorithms via random offspring generation. *Multiple Approaches to Intelligent Systems*, pages 127--136, 2004.

[SBM05] K.O. Stanley, B.D. Bryant, and R. Miikkulainen. Evolving neural network agents in the NERO video game. In *Proceedings of the IEEE symposium on computational intelligence and games*, pages 182--189, 2005.

[SE98] Y. Shi and R. Eberhart. Parameter selection in particle swarm optimization. In *Evolutionary Programming VII*, pages 591--600, 1998.

[TBS04] C. Thurau, C. Bauckhage, and G. Sagerer. Imitation learning at all levels of game-AI. In *Proceedings of the international conference on computer games, artificial intelligence, design and education*, pages 402--408, 2004.

[Thr95] S. Thrun. Learning to play the game of chess. *Advances in Neural Information Processing Systems*, pages 1069--1076, 1995.

[Ven02] G. Venter. Particle swarm optimization. In *AIAA journal*, 2002.

[Won08] K. Wong. Adaptive computer game system using artificial neural networks. In *Neural Information Processing*, pages 675--682, 2008.