Knowledge-Based Systems 34 (2012) 91–96

Contents lists available at SciVerse ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Game team balancing by using particle swarm optimization

Shih-Wei Fang, Sai-Keung Wong *

Department of Computer Science, National Chiao Tung University, Hsin-chu, Taiwan

ARTICLE INFO

Article history:
Available online 10 March 2012

Keywords:
Artificial neural network
Particle swarm optimization
Game balance
Role-playing game
Team balancing system

ABSTRACT

Game balancing affects the gaming experience of players in video-games. In this paper, we propose a novel system, team ability balancing system (TABS), which is developed for automatically evaluating the performance of two teams in a role-playing video game. TABS can be used for assisting game designers to improve team balance. In TABS, artificial neural network (ANN) controllers learn to play the game in an unsupervised manner and they are evolved by using particle swarm optimization. The ANN controllers control characters of the two teams to fight with each other. An evaluation method is proposed to evaluate the performance of the two teams. Based on the evaluation results, the game designers can adjust the abilities of the characters so as to achieve team balance. We demonstrate TABS for our in-house MagePowerCraft game in which each team consists of up to three characters.

1. Introduction

Keeping the power of abilities balanced among the characters of different classes, races, or allies is an important issue so as to uphold fun and fairness in a role-playing video-game. The players' gaming experiences may be affected by the fairness in the 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 have player versus player systems. To achieve such demand, it could involve tuning parameters and settings for more than hundreds of times. For some games, this may be a never-ending tuning cycle. It may consume a lot of time, human resources and money.

The typical solution to achieving game balance is to run a series of tests which are played by human or scripted artificial intelligence (AI). Some people might perform an actuarial study about the abilities of all the characters/teams to check game balance. An alternative is to use scripted AI. However, the results of the tests might be biased due to the changeless action rules defined by the scripts. It may be tedious and time consuming to write the scripts for different situations. A variety of techniques have also been developed for dynamic game difficult balancing in which the skills of agents are dynamically adjusted. Our focus is on team balancing and the game will be played by players.

Team balancing is one of the important topics in game balancing. Evaluating team balancing is complicated due to the rich combination of characters with different attributes and skills. In this paper, we present a novel *team ability balancing system* (TABS), which assists game designers in evaluating team balance automatically in a role-playing video game. TABS uses artificial neural network (ANN) controllers in conjunction with particle swarm optimization (PSO).

It is known that support vector machine using sigmoid function as its kernel function is equivalent to a two-layer, artificial perceptron neural network [1]. Genetic algorithm (GA) and PSO are population-based stochastic methods. The problems which could be solved by using GA may be solvable by using PSO. Furthermore, PSO has fewer parameters to tune. The Markov model is not suitable in our case as it is complicated to define the action rules [2]. On the other hand, the association between ANN and PSO is easy to build. Each particle of a swarm is associated with an ANN controller. Furthermore, there is a one-to-one correspondence between the coordinates of a particle and the weights of the ANN of the controller.

The performance of an ANN controller is evaluated based on a non-differentiable fitness function. PSO is suitable in our case as PSO is shown to be effective in optimizing non-differentiable functions [3]. The controllers learn the ways to assist team-mates, attack opponents and avoid damages in an unsupervised manner. The advantage of employing the unsupervised approach is that it is unnecessary to define the action rules.

2. Related work

PSO was introduced by Kennedy and Eberhart [4]. The development of PSO was inspired by the elegant motion of a flock of birds searching for food. Compared to Genetic Algorithm (GA) [5,6], PSO has no evolutionary operator [7]. An in-depth comparison between

* Corresponding author.
E-mail addresses: teaXrice@gmail.com (S.-W. Fang), cswingo@cs.nctu.edu.tw (S.-K. Wong).

0950-7051/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.knosys.2012.02.017

PSO and GA can be found in [8]. PSO has been adapted to various applications, such as clustering and classification [9], ANN controllers evolving [10], scheduling [11], multi-fault classification [12] and multi-objective optimization [13]. PSO has also been applied in tackling the job shop scheduling problem [14] and detecting the top management fraud [15], respectively. Shi and Eberhart [16] proposed a method for PSO parameter selection. The analysis of PSO and evolutionary algorithms can be found in [17].

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 or PSO. ANN controllers are evolved with PSO in a competitive approach [18]. In [19], ANN controllers are adopted in a video game. A real-time difficulty adjustment approach is developed in [20]. Thrun [21] used explanation-based neural network to play chess. Duro and Oliveira [22] applied PSO to estimate the weights of the heuristic evaluation function in playing chess.

Leigh et al. [23] adopted a coevolutionary algorithm to balance a continuous gaming environment in a real-time two-player action game. Each player has three different actions to adopt. Finite state machine is used for modelling the action behaviour of a player. We also apply a coevolutionary algorithm to evolve the controllers. However, the strategies of our game are not supplied. Such strategies subject to change if the attributes of the characters are changed. Hence, we want the controllers to learn the ways to play the game.

Dynamic game difficult balancing changes the attributes or fighting patterns of the controllers for matching the skill level of the players. Andrade et al. [2] adopted Q-learning to solve the Markov Decision Processes for computing optimal strategies offline. At runtime, an action mechanism is invoked for dynamically choosing the appropriate actions for the controllers. Lee [24] proposed a rule-based self-learning technique which allows the controller to play against itself. Some methods adopt dynamic scripting for achieving dynamic game balancing [25]. The action rules should be supplied in these approaches. In our case, the characters of both teams will be played by players. Furthermore, our system is used for evaluating team balance automatically.

3. Modelling using particle swarm optimization

We implement PSO according to the work of [26]. Let n be the number of particles of a swarm. The position of the ith particle is $x_i = (x_{i1}, x_{i2}, \ldots, x_{iD})$ and its velocity is $v_i = (v_{i1}, v_{i2}, \ldots, v_{iD})$ in a D-dimensional space. The range of x_{ij} is $[-4, 4]$ in our experiments. The position of each particle represents a possible solution of the optimization problem. Initially, the positions and velocities of the particles are randomly generated over the entire search-space. The particles are iteratively updated for refining the candidate solutions. The velocity of the ith particle is given by:

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

where the superscripts $(t+1)$ and t denote the iteration steps, p_i is the best position which the i-th particle has found so far, p_g^t is the global best position of all the particles at the tth iteration, r_1 and r_2 are the random values in $[0, 1]$; w, c_1 and c_2 are used to control the behaviour and efficacy of PSO. The position of the i-th particle is updated as follows:

$$x_i^{t+1} = x_i^t + v_i^{t+1}, \quad (2)$$

Note that if a particle moves outside of the search-space, the particle is set back to its nearest side constraint (i.e. setting to -4 or 4) and its velocity is set to zero. The objective function of PSO is the fitness function that is defined in Section 5.2. The fitness function is used to evaluate the performance of a controller and PSO is used to maximize the fitness value of the controller.

4. Team Ability Balancing System (TABS)

TABS can handle two teams fighting with each other and evaluate their performance in a role-playing video game. Fig. 1 shows the workflow of TABS. We use ANN controllers to control the behaviour of characters. The ANN controller computes the outputs which are used for regulating the behaviour of a character. A PSO manager maintains a set of particles of a swarm and each particle is associated with a character. The positions of the particles are updated according to the fitness values of the characters. The weights of an ANN are interpreted as the coordinates of a particle. The dimension of the position of a particle is the number of weights of an ANN. There are four steps to perform in TABS:

Step One: Setup ANNs and PSOs. Assume that we have m teams which are subjected for being evaluated their ability power among them. For each team, we build up its own PSO manager, create a population of n candidate controllers and a coach controller.

Step Two: Update PSOs and Weights of ANNs. Each particle of a swarm is associated with an ANN controller. Furthermore, there is a one-to-one correspondence between the coordinates of the particle and the weights of the ANN controller. The higher the fitness value of a candidate controller the better the weights of its ANN are. The current global best position p_g^t is equal to the 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 weights of its corresponding ANN which has the highest fitness value so far. The velocities and positions of all the particles are updated based on Eqs. (1) and (2). After that the new coordinate values of each particle are set as the weights of the ANN of the corresponding controller.

Step Three: Training Session. In the training session, we evolve the candidate controllers so that they can control the teams to play the game smartly according to the game rules. The n candidate controllers of a team are trained by the coach controllers of other teams (not including its own coach controller) one by one. In this way, the candidate controllers can be adapted to the playing behaviours of different teams.

During the training process, a fitness value is calculated and assigned to each candidate controller. After all the candidate controllers of all the teams have finished their games, we find out the best candidate controller of each team based on their fitness values. Then we make copies of the best m candidate controllers. After that, we replace the coach controller of each team as the copy of the best candidate controller of the team. The performance of the coach controller is getting improved gradually. This also ensures that each candidate controller of a team is trained by the coach controllers of the same level, which means that the candidate controllers are all trained in a fair manner. In this way, it is faster to

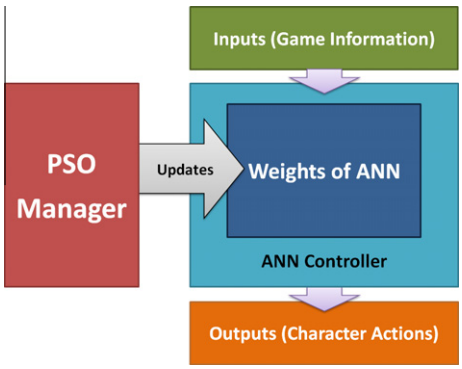

Fig. 1. The workflow of TABS.

train the candidate controllers with satisfactory performance. If the training process is done, we go to Step Four. If not, we go to Step Two.

Step Four: Playing Games by Best Trained Controllers. We use the best trained candidate controller of each team to fight with each other in a series of games and the gaming statistics are recorded. We then compute the performance scores of these candidate controllers. The ability power of the teams can be evaluated.

Parallel Training: The training computation is affected by the number of candidate controllers per character (population size), the number of characters and the complexity of the game-logic. The game-logic must be evaluated every time step during the training sessions for every controller. Therefore, parallel computation is highly demanded.

We set up game spaces which are isolated from each other. Each game space has its own dynamic data and its 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. The training session is performed in parallel by using multi-threading technology. Each game space is executed by one thread. 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. Mersenne Twister [27] is used in TABS. We need to reset all the dynamic data of the game space before a training session starts.

Game Simulation. To evaluate the fairness in a game, we ignore all the factors that are affected by human–machine interaction, such as the accuracy of aiming targets and pressing keys for certain actions (e.g. casting spells). These factors which affect the game play vary from person to person, and lead to difficulties in analysing the ability power of the teams. In TABS, we assume that a controller can perform any action accurately, such as casting spells and moving.

4.1. Inputs and outputs of ANN

In different games, players decide to perform actions according to the given information and their playing strategies; this makes a game unique. Similarly, the information which is important for making correct decisions should be fed to the controllers. This kind of information includes the current status of the characters, such as health points and the distances between the characters. The set of inputs are normalized to a unit interval [0, 1]. The outputs of an ANN are usually related to the available actions of its corresponding character. The inputs and outputs of our game, MagePowerCraft, are given in Section 5.1.

4.2. Fitness function

We evaluate the performance of a player by checking his/her accomplishment of 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. The candidate controllers with the highest fitness value are used for computing the global best positions. Hence, a fitness function affects the final behaviour of a candidate controller. A fitness function example of our game, MagePowerCraft, is given in Section 5.2.

5. Case study: MagePowerCraft

MagePowerCraft is a 3D team combating game. Fig. 2 shows the game play snapshots of MagePowerCraft. It supports up to maximum of 3-on-3 team combating inside an arena. We set a time limit for the game play. There are three classes of characters which are the Fire Wizard, the Ice Wizard and the Priest. Each class has four different skills. 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 only on either friendly or hostile targets.

There are attributes of the skills which include the power rating of the skill, the area of effect, casting time, cost of mana power (MP) per cast, the cooldown (CD) time and special effects. The settings of the skills are listed in Appendix A. The special effects of a skill may bring different de-buffing (negative) status to the targets. For examples, burning keeps on reducing the health point (HP) of its target 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 and stunning make their targets temporarily immobilized and interrupt the spells being cast by the targets.

Except for the skill settings, all other attributes such as total HP, total MP, min/max magical attack power of the three classes are the same. In this way, we can focus on evaluating the fairness of the power of skills in MagePowerCraft. The settings of the attributes of the three classes are listed in Table 1.

Fig. 2. Game play screen shots of MageCraft, playing by ANN controllers.

Table 1
Character attributes of all classes. MAP: magical attack power.

HP	MP	Basic MP	Min. MAP	Max. MAP	Magical defense	Mana recovery rate	Moving speed
20,143	20,836	6840	1642	1715	2875	0.585%	2.3529 m/s

A skill might cause several damage hits to its targets. The actual magical attack power of a character for each hit is computed by:

$$ATK = MAP_{min} + r_3(MAP_{max} - MAP_{min}), \quad (3)$$

where MAP_{min} and MAP_{max} are the minimum and maximum of the magical attack power of a caster, and r_3 is a random value inside [0, 1]. The amount of damage per hit is given by:

$$Damage = max(0, ATK \times SkillPower + ExtraDmgPt - DefensePt), \quad (4)$$

where *SkillPower* and *ExtraDmgPt* are the parameters of the skill, and *DefensePt* is the current defense power of the target.

5.1. ANN controller settings

The ANN inputs are: the ready-rate $Ready_k$ of the kth skill, the health point, the remaining time duration of each de-buffing status of each hostile character, the distance between two characters, the casting skill of a character. $Ready_k$ is computed as $Ready_k = (TotalCD_k - RemainCD_k)/TotalCD_k$, where $TotalCD_k$ is the total CD time of $skill_k$ and $RemainCD_k$ is the remain CD time of $skill_k$, for $k = 1, 2, \ldots,$ *Total Skill Number*. All the input values are normalized to the interval [0, 1]. For examples, the health point is normalized by the maximum health points, the remaining time for de-buffing is normalized by the maximum de-buffing duration, and the distance is normalized by the length of the arena floor. A casting skill is mapped to a number so that the casting skill is normalized by the total number of skills.

The five outputs of an ANN are: (1) the friendly (including self-targeting) character target; (2) the hostile character target; (3) the movement target; (4) a skill; if an aiding skill is indicated, then the skill is cast to the friendly target; if the skill is an attacking skill, it is cast to the chosen hostile target; and (5) the movement type, i.e. stop moving or moving toward/away from the movement target.

We classify the skills into two types: skills cast on friendly targets and skills cast on hostile 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. Fitness function

We design a fitness function for evaluating the performance of the controllers. The total fitness $F(t + 1)$ at the $(t + 1)$th step is given by:

$$F(t+1) = F(t) + max(0, F_{gain} - F_{loss}), \quad (5)$$

where t is the game step, $F(t)$ is the fitness value at the tth step, F_{gain} and F_{loss} are the gain and loss values, respectively. $F(0)$ is equal to 0. The fitness function is not differentiable due to the discrete nature.

F_{gain} is the sum of the following six items: the damage inflicted to the hostile targets; the healing point done to/from the friendly targets; and the four rewards. The rewards include: starting to cast spells, casting spells successfully, interrupting the spells being cast by hostile targets, and moving away from the attack range of the spells of the hostile targets.

The fitness value is decreased by F_{loss} which is the sum of the following four items: the damage inflicted by the hostile targets; the punishment for casting spells in CD; the punishment for casting spells on the invalid target (out of range); and the punishment for being interrupted while casting spells.

5.3. Experiments and results

We implemented TABS and applied it to MagePowerCraft. We performed three experiments in which two teams were fighting against each other. All experiments were performed on Quad Core CPU – Intel (R) Core (TM) i7 CPU 870 @ 2.93 GHz and the OS was Windows 7 Professional. Four threads were used for performing the training sessions.

The number of neurons in the hidden layer should be between the size of the input layer and the size of the output layer [28]. In our system, the ANN of each controller has one hidden layer consisting of six neurons. The sigmoid function is adopted as the activation function.

The winning condition of a team is that it defeats the other team before the game play duration is over. The score of a team is calculated as follows: the winning team is rewarded with a score of one; if no team is defeated, the game is drawn and each team is rewarded with a score of one.

The PSO parameters w,c_1, and c_2 were set to 0.5, 1.5 and 1.5 in all the experiments, respectively. We also performed experiments with other sets of values, as shown in Table 2. The average number of convergence iterations ranges from 56.2 to 85.9. The scores of the teams were similar in different tests with different parameters.

There were 100 candidate controllers for each class of characters. After the training sessions, we applied the best-trained controllers to control the characters to play the game. The performances of the well-developed controllers are quite good. We observe some action patterns performed by the characters, such as attacking enemies, aiding teammates, dodging attack, pursue and fleeing, in a reasonable way. However, the scores of the controllers vary from game to game. Hence, we collected the results of ten runs for each experiment and take the average of the results. There were 1000 games to play for each run and the statistics of the combating result were recorded. The average error = $|S - N/2|/(N/2)$, where S is the average score of the 10 runs and $N = 1000$.

Experiment One: We validated TABS in three cases which had different settings. We assigned the characters of the same classes to each team. Obviously, the ability power of the two teams was balanced. Thus, the same conclusion should be drawn by TABS. The two teams in the three cases are: (Case 1a) Fire Wizard (Team 1) versus Fire Wizard (Team 2), (Case 1b) Fire Wizard and Ice Wizard (Team 1) versus Fire Wizard and Ice Wizard (Team 2) and (Case 1c) Fire Wizard, Ice Wizard and Priest (Team 1) versus Fire Wizard, Ice Wizard and Priest (Team 2). Table 3 shows the statistics of the three cases. Table 4 shows the training time per game and ANN configuration.

Table 2
Number of convergence iterations for different PSO settings.

PSO weights	# Convergence iterations										Average
$\{w = 0.4, c_1 = 1.5, c_2 = 2\}$	62	67	65	60	80	70	42	69	56	55	62.6
$\{w = 0.5, c_1 = 1.5, c_2 = 1.5\}$	82	67	65	42	68	50	41	40	41	66	56.2
$\{w = 0.5, c_1 = 1.5, c_2 = 1.5\}$	70	97	96	226	67	53	57	69	40	70	82.5
$\{w = 1.4, c_1 = 1.5, c_2 = 2.5\}$	101	64	120	181	73	80	41	55	71	73	85.9
$\{w = 0.9, c_1 = 1.75, c_2 = 3\}$	141	86	58	41	49	82	71	49	60	111	74.8

Table 3
Results of experiment one.

Experiment 1a. 1-on-1 Combating					
Team1:Team2	480:520	498:502	503:497	482:518	500:500
Team1:Team2	507:493	493:507	502:498	498:502	499:501
Average	496.2:503.8		Avg. Error	0.76%	
Experiment 1b. 2-on-2 Combating					
Team1:Team2	504:496	515:485	505:495	500:500	501:499
Team1:Team2	507:493	498:502	497:503	523:477	508:492
Average	505.8:494.2		Avg. Error	1.16%	
Experiment 1c. 3-on-3 Combating					
Team1:Team2	477:523	497:503	493:507	590:410	532:468
Team1:Team2	501:499	498:502	487:513	495:505	521:479
Average	509.1:490.9		Avg. Error	1.82%	

Table 4
Results of experiment one: training time per game (s) and ANN configuration.

Experiments	Training time per game (s)	#Input	#Outputs	#Weights
1-on-1	1.42	11	5	96
2-on-2	7.27	19	5	144
3-on-3	19.44	27	5	192

The results show the largest average error is 1.82% and we use this error, namely ϵ, for determining whether or not the ability power of two teams are similar. If the error of a testing result is lower than ϵ, the ability power of the two teams is balanced. Notice that ϵ is subjected to change for different games with different features.

Experiment Two: The first team has a Fire Wizard and a Priest while the second team has an Ice Wizard and a Priest. The statistics are shown in Table 5. We assume that a game designer wants to balance two teams.

The error shows that the two teams were not balanced ($7.6\% > \epsilon$). Team 1 was more powerful than Team 2. We strengthened the Ice Wizard in Team 2 by setting the attack power of *Blizzard* from $ATK \times 210\% + 334$ to $ATK \times 265\% + 385$, and *Instant Freeze* from $ATK \times 226\% + 270$ to $ATK \times 290\% + 410$. Then we applied TABS for the new skill settings and repeated the experiment. The two teams are now balanced as the average error is 0.64%, as indicated in Table 6.

Experiment Three: The first team has an Ice Wizard and a Fire Wizard, while the second team has an Ice Wizard and a Priest. The statistics are shown in Table 7.

The error shows that the two teams were not balanced ($3.08\% > \epsilon$). Team 1 was more powerful than team 2. We weakened the ability of Fire Wizard in Team 1 by lowering the attack power of *Fire Ball* from $ATK \times 512\% + 710$ to $ATK \times 350\% + 571$, the attack power of *Fire Wall* from $ATK \times 184\% + 162$ to $ATK \times 180\% + 162$, and the attack power of *Fire Shots* from $ATK \times 211\% + 134$ to $ATK \times 185\% + 134$. Then we applied TABS for the new skill setting and repeated the experiment. The two teams are balanced as the average error is 0.42%, as indicated in Table 8.

Table 5
Results of experiment two before balancing (using original skill settings).

Team1:Team2	546:454	549:451	514:486	510:490	542:458
Team1:Team2	512:488	544:456	581:419	559:441	523:477
Average	538:462		Avg. Error	7.6%	

Table 6
Results of experiment two after balancing (with the new skill settings).

Team1:Team2	496:504	503:497	512:488	516:484	502:498
Team1:Team2	506:494	497:503	493:507	509:491	498:502
Average	503.2:496.8		Avg. Error	0.64%	

Table 7
Results of experiment three before balancing (using original skill settings).

Team1:Team2	537:463	564:436	489:511	503:497	516:484
Team1:Team2	513:487	518:482	524:476	495:505	495:505
Average	515.4:484.6		Avg. Error	3.08%	

Table 8
Results of experiment three after balancing (with the new skill settings).

Team1:Team2	518:482	505:495	503:497	520:480	478:522
Team1:Team2	493:507	497:503	501:499	494:506	470:530
Average	497.9:502.1		Avg. Error	0.42%	

Comparison to GA. We implemented GA for evolving the weights of ANN controllers. The training times per game is similar to the ones of using PSO. Most of the computation time was spent in performing the game simulation in both methods. We did not observe much difference for the fighting patterns of the characters.

6. Discussions

When the power of the two teams is unbalanced, we tune the subjected parameters up or down a bit according to the statistics of the two teams given by TABS. After each tuning attempt, we evaluate the balance situation of the two teams with the new setting. The process of tuning the subjected parameters is required manual operation. It usually takes two to four attempts.

Currently, a team is scored by its win-loss record. The criteria for computing the score of a team can be extended to other aspects. For example, a character may be easily killed at the very beginning of a game. We believe that this is not a good gaming experience for the player who controls that character. In this case, TABS can show the information, such as the average surviving time and average score of each character. This kind of information may be useful for intra-team balancing.

7. Conclusions and future work

In this paper, we introduce Team Ability Balancing System (TABS) which evaluates automatically the fairness of teams in a role-playing video game. The statistics collected by TABS are useful for game designers to tune the gaming parameters. We have validated TABS by applying it to our in-house MagePowerCraft. It is an interesting research direction to find a better and promising solution for verifying the maturity of the controllers automatically. Currently, we require game designers to observe how the controllers play in order to determine whether or not the controllers play in a reasonable way. We want to develop a fully automatic evaluation system for fairness adjustment in team combating games.

Appendix A. Skill settings in MagePowerCraft

See Tables A.9 and A.10.

Table A.9
Skill settings. The actual attack power $(alpha, \beta) = ATK \times \alpha + \beta$, where ATK is calculated as shown in Eq. (3). Heal Points = Target's maximum $HP \times 8\% + 324$.

Characters	Skills	Attack power	Heal	MP cost (%)	Target number	Attack range (m)	Max damage	Cast time (s)	CD time (s)	Area effect	Area effect width (m)
Fire wizard	Fire ball	(512, 710)	–	3.4	Single	0–10.59	1	2	17	–	–
	Inferno	(185, 276)	–	3.4	Multiple	0.5–3.53	12	0.8	24	Rectangular	0.6
	Fire wall	(184, 162)	–	3.2	Multiple	0–2.59	6	1.2	33	Circular	–
	Fire shots	(211, 134)	–	2.5	multiple	0–10.59	3	0.1	20	Rectangular	0.6
Ice wizard	Freezing sword	(214, 556)	–	2.3	multiple	0–2.35	3	0.1	15	Rectangular	0.6
	Freezing field	(279, 462)	–	3.1	multiple	2.59–5.41	1	1.1	28	Circular	–
	Blizzard	(210, 334)	–	2.3	multiple	0.6–2.59	2	0.9	15	Circular	–
	Instance freeze	(226, 270)	–	2.3	multiple	0–1.88	1	0.5	24	Circular	–
Priest	Chain lightning	(220, 85)	–	2.5	multiple	0–5.89	3	0.8	20	Circular	–
	Grand cross	(169, 262)	–	3.2	multiple	0.5–11.29	6	2.6	28	Rectangular	1.8
	Heal	–	(8, 324)	3	multiple	0–4	1	0.5	60	Circular	–
	Lightning volt	(181, 294)	–	2.7	multiple	0–4.94	5	0.5	18	Rectangular	0.6

Table A.10
Special effects of skills.

Characters	Skills	Special effect
Fire wizard	Fire ball	Stunned, last for 1.5 s. Burned, take 10% of last hit damage point every 2 s. and last for 8 s
	Inferno	Burned, take 50% of last hit damage point every 2 s and last for 10 s; Stunned, last for 4 s
	Fire wall	Burned, take 5% of last hit damage point every 2 s and last for 10 s; Stunned, last for 3 s
	Fire shots	Burned, take 10% of last hit damage point every 2 s and last for 15 s
Ice wizard	Freezing sword	Frostbitten, *Moving speed* × 50% and last for 2 s; Stunned, last for 3.5 s
	Freezing field	Frostbitten, *Moving speed* × 50% and last for 8 s
	Blizzard	Frostbitten, *Moving speed* × 50% and last for 1 s; Stunned, last for 0.5 s
	Instance freeze	Frozen, last for 2 s
Priest	Chain lightning	Electrocuted, break actions every 5 s and last for 12 s
	Grand cross	Stunned, last for 1 s
	Lightning volt	60% Electrocuted, break actions every 5 s and last for 13 s

References

[1] P. Andras, The equivalence of support vector machines and regularization neural networks, in: Neural Processing Letters, vol. 65, 2002, pp. 97–104.
[2] G. Andrade, G. Ramalho, H. Santana, V. Corruble, Challenge-sensitive action selection: an application to game balancing, in: Proceedings of the Int'l Conference on Intelligent Agent Technology, 2005, pp. 194–200.
[3] X. Yu, X. Xiong, Y. Wu, A pso-based approach to optimal capacitor placement with harmonic distortion consideration, Electric Power Systems Research 71 (2004) 27–33.
[4] J. Kennedy, R. Eberhart, Particle swarm optimization, in: Neural Networks, vol. 4, 1995, pp. 1942–1948.
[5] D. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.
[6] J. Branke, Evolutionary algorithms for neural network design and training, in: Nordic Workshop on Genetic Algorithms and its Applications, 1995.
[7] R. Eberhart, Y. Shi, Comparison between genetic algorithms and particle swarm optimization, in: Evolutionary Programming VII, 1998, pp. 611–616.
[8] R. Hassan, B. Cohanim, O. De Weck, G. Venter, A comparison of particle swarm optimization and the genetic algorithm, in: AIAA Multidisciplinary Design Optimization Specialist Conference, 2005, pp. 1–13.
[9] K.Y. Huang, A hybrid particle swarm optimization approach for clustering and classification of datasets, Knowledge-Based Systems 24 (2011) 420–426.
[10] R. Eberhart, Y. Shi, Particle swarm optimization: developments, applications and resources, in: Evolutionary Computation, vol. 1, 2001, pp. 81–86.
[11] B. Alatas, E. Akin, Multi-objective rule mining using a chaotic particle swarm optimization algorithm, Knowledge-Based Systems 22 (2009) 455–460.
[12] X. Tang, L. Zhuang, J. Cai, C. Li, Multi-fault classification based on support vector machine trained by chaos particle swarm optimization, Knowledge-Based Systems 23 (2010) 486–490.
[13] S. Qasema, S. Shamsuddina, A. Zain, Multi-objective hybrid evolutionary algorithms for radial basis function neural network design, Knowledge-Based Systems 27 (2012) 475–497.
[14] R. Zhang, S. Song, C. Wu, A two-stage hybrid particle swarm optimization algorithm for the stochastic job shop scheduling problem, Knowledge-Based Systems 27 (2012) 393–406.
[15] P.-F. Paia, M.-F. Hsub, M.-C. Wang, A support vector machine-based model for detecting top management fraud, Knowledge-Based Systems 24 (2012) 314–321.
[16] Y. Shi, R. Eberhart, Parameter selection in particle swarm optimization, in: Evolutionary Programming VII, 1998, pp. 591–600.
[17] P. Angeline, Evolutionary optimization versus particle swarm optimization: philosophy and performance differences, in: Evolutionary Programming VII, 1998, pp. 601–610.
[18] L. Messerschmidt, A. Engelbrecht, Learning to play games using a pso-based competitive learning approach, IEEE Transactions on Evolutionary Computation 8 (2004) 280–288.
[19] K. Stanley, B. Bryant, R. Miikkulainen, Evolving neural network agents in the nero video game, in: Symposium on Computational Intelligence and Games, 2005, pp. 182–189.
[20] J. Olesen, G. Yannakakis, J. Hallam, Real-time challenge balance in an RTS game using rtneat, in: Computational Intelligence and Games, 2008, pp. 87–94.
[21] S. Thrun, Learning to play the game of chess, Advances in Neural Information Processing Systems (1995) 1069–1076.
[22] J. Duro, J. d. Oliveira, Particle swarm optimization applied to the chess game, in: IEEE Congress on Evolutionary Computation, 2008, pp. 3702–3709.
[23] R. Leigh, J. Schonfeld, S. Louis, Using coevolution to understand and validate game balance in continuous games, in: Genetic and evolutionary computation, 2008, pp. 1563–1570.
[24] C. Lee, A self learning rule-based controller employing approximate reasoning and neural net concepts, International Journal of Intelligent Systems 6 (1) (1991) 71–93.
[25] P. Spronck, I. Sprinkhuizen-Kuyper, E. Postma, Difficulty scaling of game ai, in: Proceedings of the 5th International Conference on Intelligent Games and Simulation, Belgium, 2004, pp. 33–37.
[26] G. Venter, Particle swarm optimization, AIAA Journal (2003) 1583–1589.
[27] M. Matsumoto, T. Nishimura, Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator, ACM Transactions on Modeling and Computer Simulation 8 (1998) 3–30.
[28] M. Berry, G. Linoff, Data Mining Techniques for Marketing, Sales, and Customer Support, Wiley, New York, 1997.