Emergent Tactical Formation Using Genetic Algorithm in Real-Time Strategy Games

Chih-Sheng Lin and Chuan-Kang Ting
Department of Computer Science and Information Engineering
National Chung Cheng University, Chia-Yi 621, Taiwan
Email: {lcs99m, ckting}@cs.ccu.edu.tw

***Abstract*—Current tactical formations used in real-time strategy games are mainly fixed. However, fixed formations are incapable of adaptation to unforeseen game situations. This paper proposes the emergent tactical formation based on genetic algorithm (GA) to address this issue. The proposed method first parameterizes tactical formations and then uses GA to optimize the parameters. The experimental results on StarCraft testbed show that our emergent tactical formation can defeat the intrinsic game AI in the testbed of small flat-terrain environment, which validates the effectiveness of the proposed method.**

***Index Terms*—Genetic algorithm; tactical formation; real-time strategy games.**

I. Introduction

Developing intelligent human-like non-player characters is a key task in modern game design. Particularly, in real-time strategy (RTS) games, first-person shooter (FPS) games, and massive multiplayer online role-playing games (MMORPG), the hostile non-player characters must be able to provide enough challenges in order to attract players to defeat the opponents. On the other hand, the non-player characters as wingmen must smartly assist players in accomplishing missions. Design of intelligent non-player characters can be classified into two mechanisms by the way of behavior-making: agent-based AI and non-agent-based AI [1]. In the agent-based AI, the non-player characters behave depending on the current game environment, e.g., other characters nearby. By contrast, in the non-agent-based AI the behavior of all characters is controlled by a top-down system. These two mechanisms can be used separately or jointly in the game AI.

Beyond individual behavior, many games need a good design of AI for controlling groups of characters. For example, it should achieve a natural and reasonable moving behavior for a troop when the player commands; meanwhile, the characters should coordinate their behavior as a team. Such a united behavior requires the tactical formation, which is ordinarily used in real-world military tactics. The tactical formations for games are mostly generated by a static strategy. However, this static method cannot afford to adapt the formation to unforeseen situations. To address this issue, van der Heijden et al. [2] proposed dynamic tactical formation; nevertheless, this method is still based on fixed formations.

This study presents a more flexible and robust method based on genetic algorithm (GA) to form the formations that are capable of adaptation to unforeseen game situations. The proposed emergent formation method belongs to agent-based AI. More specifically, we parameterize the tactical formations and then use GA to optimize these parameters. The remainder of this paper is organized as follows. Section II describes the background knowledge about tactical formations. Section III elucidates the parameterization of emergent formations and the proposed GA. The experimental results and discussion are presented in Section IV. Finally, Section V gives the conclusions of this study.

II. Tactical Formation

A tactical formation arranges the positions of military units. In many cultures, formations are frequently used in battles, e.g., the formation of shield wall, of which each military unit holds its shield and stands shoulder to shoulder, is often used as a defensive formation in ancient wars of China. Many different types of formations have been developed in military tactics. For instance, the column formation was used in the early stages of the French Revolutionary Wars, the line formation was used in the Seven Years' War, the square formation was used in Napoleonic Wars, and the vee formation used on military flight missions [3].

In the research of game AI, the idea of using the real-world tactical formations emerged and was applied to the formation of moving a group of characters in RTS games, e.g., in Age of Empires II. The formations can be categorized into fixed formation, dynamic formation, and emergent formation [1]. The following subsections describe these three formations in more detail.

A. Fixed Formation

Fixed formations use a set of slots to arrange the positions of characters in a fixed geometric shape. A simple way is to employ a slot representing the leader of the group and the remaining slots for its followers. Note that the followers should keep a fixed distance and the same direction. Figure 1 illustrates three fixed formations, where the red slot represents the leader while the blue slots represent the followers. The followers should keep distance and direction from their leader when the leader moves toward a new position. Note that the shape formed by the slots should stay invariant for the fixed formations.

Although fixed formations are easy to implement, they lack the adaptability to new environment. This drawback is

978-0-7695-4601-8/11 $26.00 © 2011 IEEE
DOI 10.1109/TAAI.2011.63

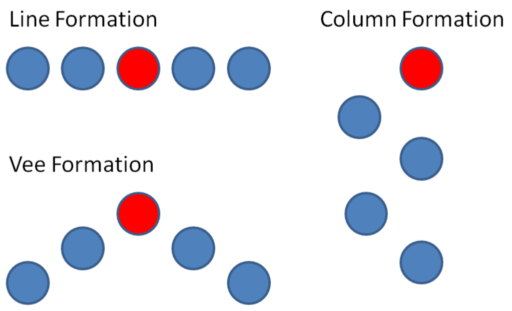


Figure 1. Fixed formations.

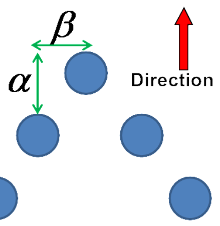


Figure 2. Parameters for dynamic formation.

particularly serious in games since there exist unforeseen situations. For example, a fixed formation is unable to deal with the case that some characters cannot fit in the shape due to different moving speeds or obstacles.

B. Dynamic Formation

The study [2] presents the dynamic formations for RTS games. This method uses some parameters to define the shape of a formation. In the example of Fig. 2, the parameters α and β specify the vertical and horizontal distances between characters in a line formation, respectively. Additional parameters are used to determine the number of line formations, the distance between different line formations, the number of overall formations composed by several line formations, and the number of units in a line formation. These parameters define the shape of formations for moving or fighting. Furthermore, this method uses a stochastic algorithm to optimize the above parameters for the best formations that can fight a certain opponent off. Once confronted by the same opponents, the game can adopt the trained parameters to combat with. As a result, the formations can change dynamically according to the type of opponents that are learned offline.

In dynamic formations, the shape of a formation is still fixed and its capability is subject to the results of offline learning. Thus, it still cannot handle the game situations not existing in the learning phase. In the light of adaptability, this study proposes using emergent formation to address this issue at dynamic formations.

C. Emergent Formation

Emergent formations do not need to fix the slots for a certain shape. That is, each character does not follow a single virtual character; instead, its position is determined according to its own steering rules, which can be based on the information from its positions or neighbors. As Fig. 3 demonstrates, the vee formation is given by two simple steering rules for each character: 1) the character selects its move target only behind its front neighbor; 2) the character should choose left side or right side of its front neighbor. In this way, the characters will organize themselves to form a small vee formation gradually.

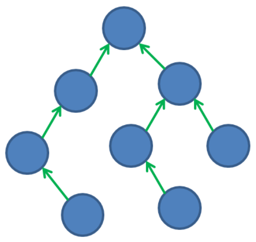


Figure 3. Emergent vee formation.

Emergent formations can be easily implemented through the steering rules. However, to find the steering rules are difficult owing to the complex interaction between characters. In addition, it is hard to analyze and control the emergent formations because the behavior of characters depends upon the present game situation such as the map and distribution of neighbors.

III. Proposed Method

A. Steering Rules for Emergent Formation

This study develops the steering rules based on the rules for boids proposed by Reynolds [4]. More specifically, we use three vectors (separation, cohesion, and alignment) to represent the steering rules. Let p_c and v_c be the position and velocity of a character c, respectively. The following introduces the three vectors.

- Separation: A character c tries to keep a small distance from its neighboring characters. Let N be the set of neighbors of c. The separation vector is defined by

$$\Delta_{\text{sep}}(c) = -\sum_{i \in N} (p_i - p_c).$$

- Cohesion: A character c tries to move toward the centroid $\bar{p}$ of its neighboring characters. The cohesion vector is defined by

$$\Delta_{\text{coh}}(c) = \bar{p} - p_c \quad \text{with} \quad \bar{p} = \frac{1}{|N|} \sum_{i \in N} p_i.$$

- Alignment: A character c tries to move towards the average direction $\bar{v}$ of its neighboring characters. The alignment vector is defined by

$$\Delta_{\text{ali}}(c) = \bar{v} - v_c \quad \text{with} \quad \bar{v} = \frac{1}{|N|} \sum_{i \in N} v_i.$$

The final direction of a character c can be calculated using the weighted sum of the above three rules:

$$\Delta(c) = w_{\text{sep}}\Delta_{\text{sep}}(c) + w_{\text{coh}}\Delta_{\text{coh}}(c) + w_{\text{ali}}\Delta_{\text{ali}}(c) \ ,$$

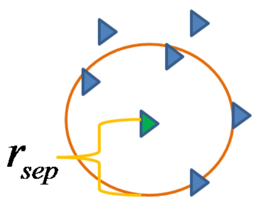


Figure 4. Range for separation.

where $\Delta(c)$ is the target vector used to calculate the positions for character c to move toward; w_{sep}, w_{coh}, w_{ali} are the weights for separation, cohesion, and alignment vectors, respectively. The target position p'_c for the movement of the character c can then be calculated by

$$p'_c = p_c + \Delta(c).$$

This study presents four steering rules for emergent formation based on Reynolds' rules. Note that our steering rules for alignment do not follow boid rules since the direction of characters should be always toward the positions of opponents. Aside from alignment, each character will move to the target position according to the following rules.

1) Move toward the position of enemy: Here we assume that the information of hostile characters such as their positions is known[1]. Using this knowledge, Rule 1 moves a character toward its nearest hostile character; that is,

$$\Delta_{r1}(c) = p_h - p_c,$$

where Δ_{r1} is the return vector of Rule 1 and p_h is the position of the nearest hostile character of c. This rule leads our characters to the enemy.

2) Keep a small distance from other friendly characters: This rule resembles Reynolds' separation rule; nonetheless, it considers only the characters within range r_{sep} (see Fig. 4). All characters must follow this separation rules beyond other rules to avoid collision while moving.

3) Move toward the centroid of a specific column formation: This rule is based on the concept of column formation, which is widely used in military tactics. As Fig. 5 shows, the column formation consists of several files and each length of the files is ordinarily longer than the length of ranks, i.e., the number of characters in a file is larger than the number of characters in a rank [6].

Let r_{col} denote the width of a column formation and r_{sig} the range of neighborhood for the current character. Here we take only the characters locating in the range of r_{sig} into account. The width r_{col} is used as a scanning window to examine the current character's neighborhood (cf. Fig. 6); meanwhile, it tries to find the best column area that achieves

$$\max_{s \in S_j} \left(\frac{|S_j|}{\|p_c + \Delta_{\text{coh}}(s)\|} \right),$$

[1]This can be achieved by enabling the `CompleteMapInformation` flag in StarCraft platform [5].

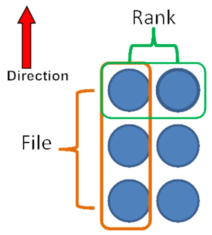


Figure 5. Column formation.

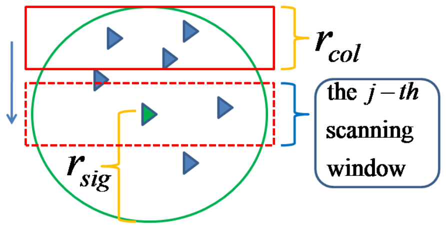


Figure 6. Finding the best column area.

where S_j is the set of characters in the j_{th} scanning window and $\Delta_{\text{coh}}(s)$ is the cohesion vector of character s. This rule moves the current character toward the area that contains most characters nearby. Once the area is confirmed, it will add a separation vector for character c in order to diversify the resultant formations. Finally, the sum of the cohesion vector and the separation vector are returned as the result of this rule.

4) Move toward the centroid of a specific line formation: This rule is based on the concept of line formation, one of the most common formations in military tactics. A line formation consists of several ranks, in which each length is usually longer than the length of files, i.e., the number of characters in a rank is larger than the number of characters in a file (cf. Fig. 7).

Line formation requires a simple rule to form: Let r_{lin} be as the height of a line formation. As Fig. 8 shows, we use the width r_{lin} as a window to scan the current character's neighborhood for the best line area as

$$\max_{s \in S_j} \left(\frac{|S_j|}{\|p_c + \Delta_{\text{coh}}(s)\|} \right),$$

where S_j is the set of characters which locate in the j_{th} scanning window. Similar to Rule 3, Rule 4 is defined to move the current character toward the area that contains most characters nearby. Then we add a separation vector to character c. The cohesion vector and the separation vector render as the final result of this rule.

With the results from the above four rules, the target position of the current character c can be obtained by

$$p'_c = p_c + w_{r1}\Delta_{r1} + w_{r3}\Delta_{r3} + w_{r4}\Delta_{r4}$$

or

$$p'_c = p_c + w_{r2}\Delta_{r2},$$

where Δ_{r1}, Δ_{r2}, Δ_{r3}, Δ_{r4} and w_{r1}, w_{r2}, w_{r3}, w_{r4} represent the vectors of above rules and the associated weights, respectively.

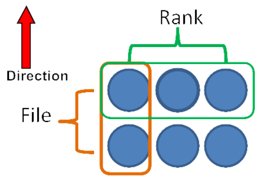


Figure 7. Line formation.

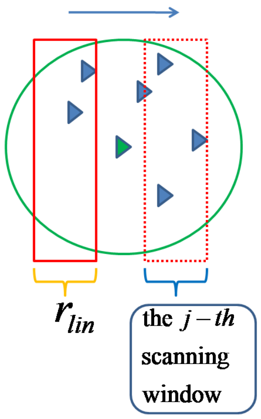


Figure 8. Finding the best line area.

Table I
PARAMETERS OF STEERING BEHAVIOR FOR OPTIMIZATION.

Parameter	Description
1. r_{sig}	Range of neighborhood of current character.
2. r_{sep}	Range for separation.
3. r_{col}	Width of column formation.
4. $r_{\text{col,sep}}$	Range for separation in column formation.
5. r_{lin}	Height of line formation.
6. $r_{\text{lin,sep}}$	Range for separation in line formation.
7. $w_{r1,a}$	Weight for Rule 1 when doesn't see the opponent yet.
8. $w_{r1,b}$	Weight for Rule 1 when see the opponent.
9. w_{r2}	Weight for Rule 2.
10. w_{r3}	Weight for Rule 3.
11. w_{r4}	Weight for Rule 4.

Table II
PARAMETER SETTING FOR THE GA.

Parameter	Value
Representation	Real-coded
Chromosomal Length	11
Population	Generational (size 20)
Parent Selection	Binary Tournament Selection
Crossover	Arithmetic ($\alpha = 0.6, r_c = 0.9$)
Mutation	Creep ($1 \leq \lvert\delta\rvert \leq 50, r_m = \frac{2}{l}$)
Survivor Selection	$(\mu + \lambda)$
Termination	100 generations
Number of Trials	24 times

B. Genetic Algorithm for Formation Optimization

To optimize the formations, we first define parameters to control the steering rules of characters for emergence of tactical formation against the enemy. Table I summarizes the parameters to be optimized. The parameters for the steering rules are encoded as genes with real values in GA, where the ranges $r_{\text{sig}}, r_{\text{sep}}, r_{\text{col}}, r_{\text{col,sep}}, r_{\text{lin}}, r_{\text{lin,sep}} \in [0,1]$ and the rule weights $w_{r1,a}, w_{r1,b}, w_{r2}, w_{r3}, w_{r4} \in [0,2]$.

When facing an unknown enemy, nevertheless, one can hardly know the correct responsive behavior for the characters. Therefore, we develop an evolutionary approach based on GA to evolve the parameters in the steering rules for fight with the enemy [7]. Table II lists the setting for the operators and parameters in the GA. The fitness value of a chromosome is determined by the total HPs of all remaining characters after a force is defeated, while the value is negative if defeated. Note that the fitness value of a chromosome is computed whenever a game is finished, i.e., some force is defeated. Due to the high computation cost, each chromosome is evaluated only once, namely by a single game simulation. Additionally, the population size is rather small in order to reduce simulation time. Through the evolutionary process of GA, the parameters of steering rules for formations are expected to get improved toward the optima.

Algorithm 1 further presents the way to control a character to fight against the game AI of StarCraft. The characters will be controlled by our defined steering rules except combat, for which the intrinsic combat strategies in StarCraft is executed instead.

IV. EXPERIMENTS

This study conducts experiments to assess the effectiveness of the proposed method. In this section, we first describe the game environment of our experiment; then we present and discuss the experimental results.

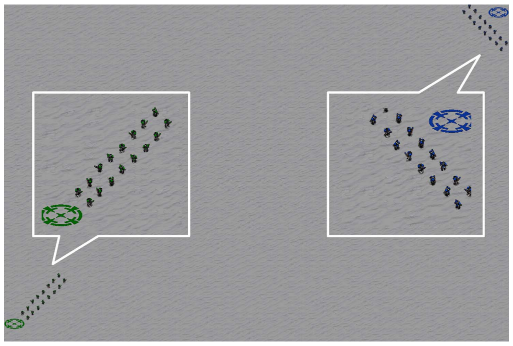
A. Game Environment

This study uses the StarCraft game environment to test our method for tactical formation. StarCraft is one of the most popular RTS games and known for its balance between the overall strength of three different races [8]. StarCraft provides the API, called Broodwar API, which allows researchers to develop their custom AI [5]. The interface gives the information about the current game situation and command units, e.g., moving, attacking, and building. StarCraft has been popularly adopted as the platform and testbed for development of game AI [9].

In this study, we modify the first tournament of StarCraft AI competition [9] as the experiment environment, in which the objective is to manage characters to destroy their enemies in the flat-terrain environment. As Fig. 9 shows, the experiment includes two maps that differ in the formations of hostile characters, i.e., line and vee formations. The number of characters is increased from 12 to 16 for each force and the type of characters is changed from Dragoon (of Pross) to Marine (of Terrans). The increase in the number of characters enables more different shapes for formation, while the change of character types reduces the running time and accelerates the simulation.

Algorithm 1 Control over a character within a time frame.

```
1) Initialize the parameters of steering rules with the results from GA.
2) If any hostile character is in the sight range of current character, let flag readyToAttack be true; otherwise false.
3) If the flag readyToAttack is true, get the target position calculated from its steering rules; otherwise go to Step 5.
4) If the distance between current position and target position is greater than the maximum range of its weapon, move the current character to the position that can attack the specified target and then go to Step 6.
5) If the current character is not moving, get the target position calculated from its steering rules and order the current character to attack the specified target.
6) Action for this time frame is completed.
```


(a) Map1

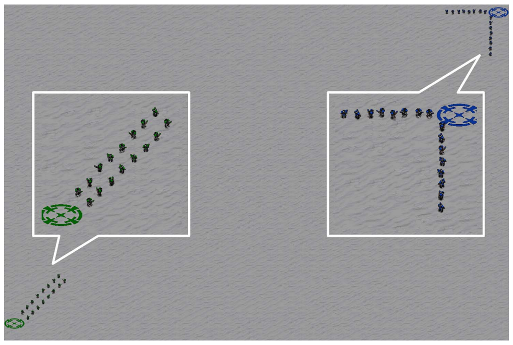
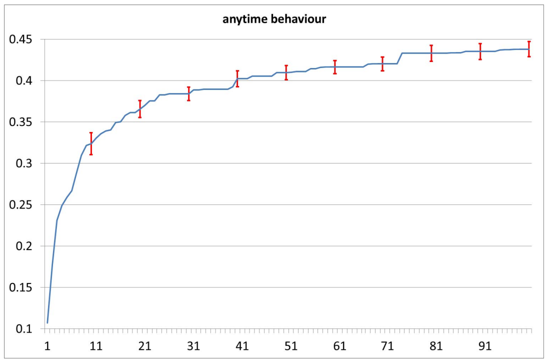

(b) Map2

Figure 9. Two maps for the experiment. The green units are our control characters and the blue units are the hostile characters.

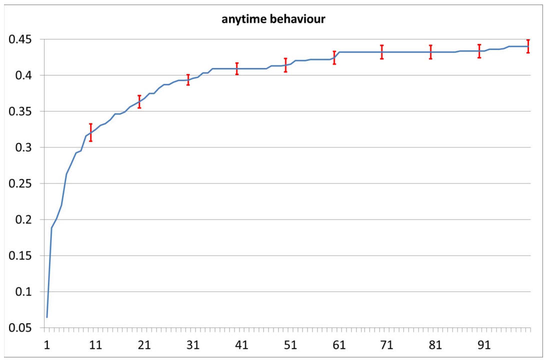
As for the design of game AI in StarCraft, one can order the characters within a time frame. The fastest rate is 23.8–24 frames per second; that is, the AI developers have only approximately 0.042 second to organize and perform the troop's action. Once unable to response on time, the characters will follow their previous order. Table III lists the functions of Broodwar API and their effects used in our experiments.

Table III
THE FUNCTIONS USED FOR THE CUSTOM GAME AI.

- `UnitType::sightRange()`
  Return how far the character type can see into the fog of war.
- `Position::getApproxDistance(Position)`
  Return the approximate distance to the given position.
- `Unit::isMoving()`
  Return true if the character is moving now.
- `Unit::attack(Position)`
  Orders the character to attack to the specified position.
- `Player::groundWeaponMaxRange(UnitType)`
  Return the maximum range of the given character type's ground weapon.



(a)



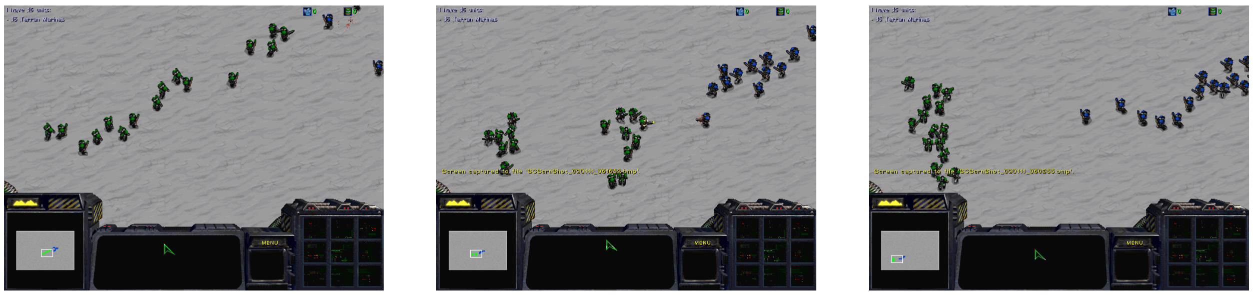
(b)

Figure 10. Progress of mean best fitness and standard error over 24 trials for (a) map1 and (b) map2.

B. Experimental Results

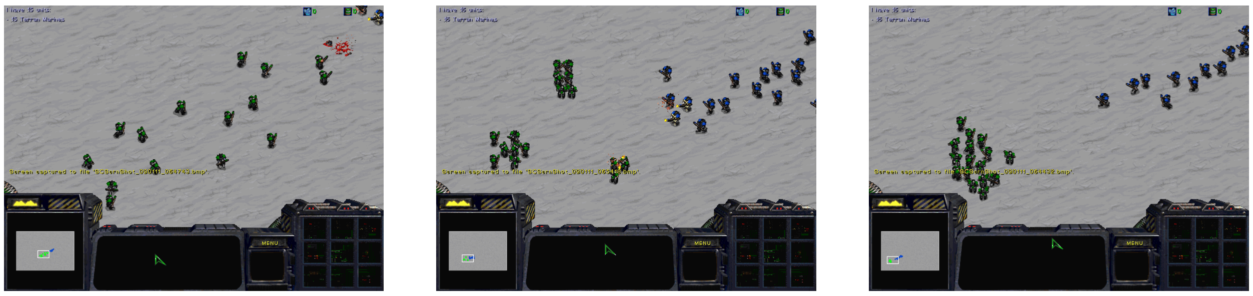
Figure 10 plots the progress of the mean best fitness and the standard error in the course of evolution. Restated, the fitness value is defined as the sum of total remaining HPs divided by the sum of total initial HPs. The positive fitness values indicate that our developed formations beat the intrinsic game AI of StarCraft. Moreover, the increase in fitness with evolution validates the effectiveness of GA in improving the formations.

Further, Figs. 11 and 12 illustrate the variation of formations at three phases of GA. The results show how GA adapts the formations to the game environment against the hostile characters. In the beginning, the formation is relatively sparse. As evolution goes, the formation density gradually increases



(a) At initialization (b) After 30 generations (c) After 100 generation

Figure 11. Formations obtained in three phases of GA for map1.



(a) At initialization (b) After 30 generations (c) After 100 generation

Figure 12. Formations obtained in three phases of GA for map2.

and forms a concentrated fire attack. The enhanced fitness value confirms the advantage of the formations obtained from GA.

V. Conclusions

This study proposes using the emergent formation to form the tactical formations in order to adapt the unforeseen environment. To address the difficulty in development and analysis of emergent formation, we parameterize the formation and then use GA to optimize the parameters for the best tactical formation. Experiments were conducted on the first-type StarCraft tournament environment. According to the experimental results, the emergent formations generated from our GA method can beat the intrinsic game AI in StarCraft in the scenario of small flat-terrain environment. The experimental results also show that GA can effectively optimize the parameters and reduce the loss of HPs for our characters.

Some directions remain for future work. First, the performance on different maps should be examined. Second, approximation of the combat behavior and simulation is needed to decrease the long time consumption in game simulation [10]. The strategy for partial observation is another important research direction.

References

[1] I. Millington and J. Funge, *Artificial intelligence for games*. Morgan Kaufmann, 2009.

[2] M. van der Heijden, S. Bakkes, and P. Spronck, "Dynamic formations in real-time strategy games," in *Proceedings on IEEE Symposium on Computational Intelligence and Games*, 2008, pp. 47–54.

[3] Wikipedia, "Tactical formation — wikipedia, the free encyclopedia," 2010. [Online]. Available: http://en.wikipedia.org/w/index.php?title=Tactical_formation

[4] C. Reynolds, "Boids," 2005. [Online]. Available: http://www.red3d.com/cwr/boids

[5] BWAPI, "The brood war application programming interface (bwapi)," 2011. [Online]. Available: http://code.google.com/p/bwapi/

[6] Wikipedia, "Column (formation) — wikipedia, the free encyclopedia," 2010. [Online]. Available: http://en.wikipedia.org/w/index.php?title=Column_(formation)

[7] D. Goldberg, *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley, 1989.

[8] Wikipedia, "Starcraft — wikipedia, the free encyclopedia," 2011. [Online]. Available: http://en.wikipedia.org/w/index.php?title=StarCraft

[9] AIIDE, "AIIDE StarCraft AI competition," 2011. [Online]. Available: http://eis.ucsc.edu/StarCraftAICompetition#StarCraftAICompetition

[10] M. Preuss, N. Beume, H. Danielsiek, T. Hein, B. Naujoks, N. Piatkowski, R. Stüer, A. Thom, and S. Wessing, "Towards Intelligent Team Composition and Maneuvering in Real-Time Strategy Games," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 2, pp. 82–98, 2010.