Cooperative Reinforcement Learning for Multiple Units Combat in StarCraft

Kun Shao[†‡], Yuanheng Zhu[†‡], Dongbin Zhao[†‡]
[†]The State Key Laboratory of Management and Control for Complex Systems, Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China
[‡]University of Chinese Academy of Sciences, Beijing 100049, China
shaokun2014@ia.ac.cn, yuanheng.zhu@ia.ac.cn, dongbin.zhao@ia.ac.cn

***Abstract*—This paper presents a cooperative reinforcement learning model to tackle the problem of multiple units combat in StarCraft. We construct an efficient state representation method to break down the complexity caused by the large state and action space in combat scenario. This method takes units' state and various distance information into consideration, including the current step and the last step. To solve the problem of sparse and delayed rewards, a reward function including small intermediate rewards is introduced. This reward function helps to balance units' move and attack, and encourages our units to fight as a team. We present gradient-descent Sarsa(λ) to train the learning model, and use a neural network as the function approximator for the Q values. The experimental results presented in this paper show our controlled units can successfully learn to combat in a cooperative way, and defeat the built-in AI in a 3 Goliaths against 6 Zealots StarCraft combat scenario.**

I. Introduction

In the last few years, we have witnessed massive progresses in the field of game artificial intelligence (AI), which reach a performance that exceeds human experts in various games, including the classic Atari 2600 video games [1] [2], the challenging board games Gomoku [3] [4] and Go [5], and the imperfect information game Texas hold'em Poker [6] [7]. However, most of the games' actions are fixed and limited, and researchers only need to control a single agent in these games. In this paper, we focus on a real-time strategy (RTS) game with multiple units and large state-action space as the testbed to study the cooperative behaviors of multi-agents. RTS games are a kind of video games and players perform as many actions as they can do during the game, which are different from taking turns in board games [8]. To win the game, players need to control many kinds of units and structures to gather resources, build base, construct an army and finally destroy the enemies [9]. In addition, RTS games are usually played by multiple players that have to deal with incomplete information during the game due to units' sight range and "fog of war".

As the most popular RTS game, StarCraft: Brood War was released by Blizzard Entertainment in 1998, and had been sold more than 9.5 million sets in the following ten years[1]. StarCraft has three different but very balanced races: Terran, Protoss and Zerg, requiring various strategies and tactics [10]. As for game AI research, StarCraft provides an ideal testbed to study the control of multiple units, and an environment to define tasks of various difficulty [11]. In recent years, the study of StarCraft has been an important field of game AI research, driven by the AIIDE[2], CIG[3] and SSCAIT[4] StarCraft AI competitions. AI in StarCraft aims at solving a series of challenges, such as spatial and temporal reasoning, opponent modeling, adversarial planning and multi-agent collaboration [8]. At present, designing a game AI for the full StarCraft only based on machine learning method is impractical. Many researchers focus on micromanagement, the small scale combat as shown in Fig.1, as the first step to study multi-agent control in StarCraft [11].

Fig. 1. Example of the StarCraft small scale combat, with 3 Goliaths against 6 Zealots in jungle world terrain.

In this paper, we present a cooperative reinforcement learning algorithm to teach multiple units to combat based on our proposed efficient state representation method and reward function. The organization of the remaining paper is arranged as follows. In Section II, we describe the problem formulation and related work about StarCraft combat research. Section III presents the state representation method and network architecture, as well as the whole model. And in Section IV, we introduce the reinforcement learning background and gradient-

This work is supported by National Natural Science Foundation of China (NSFC) under Grants No.61603382, No.61573353 and No. 61533017.

[1]https://en.wikipedia.org/wiki/StarCraft

[2]http://www.cs.mun.ca/ dchurchill/starcraftaicomp/index.shtml

[3]https://cilab.sejong.ac.kr/sc_competition/

[4]http://sscaitournament.com/

descent Sarsa(λ) for multiple units combat in details. Section V presents the training details and the analysis of experimental results. In the end, we draw a conclusion and discuss the future work on StarCraft combat research.

II. Problem Formulation and Related Work

A. Problem Formulation

StarCraft small scale combat concerns the control of multiple units to navigate in highly dynamic environment and attack enemies within fire range under certain terrain conditions. The combat environment with n our units is approximated as a Markov decision process (MDP), which is described by tuples $\{s_i, a_i, r_i, s_i'\}_{i=1}^{n}$. s_i denotes the state of the current combat from the viewpoint of unit i; a_i is the action of unit i; $s_i \times a_i \rightarrow (r_i, s_i')$ denotes the transition from state s_i to state s_i' and the generated reward r_i. In order to maintain a flexible framework and allow an arbitrary number of units, we consider that all units have access to the whole state space of the current combat from its own viewpoint, and have the same action spaces. In the combat, our units are fully cooperative with each other and against the units in the enemy side. The objective of unit i is to maximize the expected discounted reward over episodes under certain policy π, as demonstrated in equation (1).

$$\max_{\pi} \mathbb{E}[\sum_{t'=t}^{T} \gamma^{t'-t} r_{t'} | s = s_t, a = a_t, \pi] \tag{1}$$

T is the terminal time, and π is the policy mapping states to actions.

B. Related Work

Game AI researchers have studied many methods for small scale combat in StarCraft. These work involves potential fields, Bayesian modeling, heuristic game-tree search, neuroevolution and reinforcement learning. Hagelbck [12] and Uriarte *et al.* [13] use potential fields by placing attracting or repelling charges at important locations in the combat map. This kind of method can handle the game's dynamic aspects and are useful for spatial navigation and obstacle avoidance. Synnaeve *et al.* propose Baysian modeling to control units locally, with higher level orders integrated as perception [14]. They use a hierarchical model to deal with incompleteness and uncertainty in micromanagement. Churchill *et al.* present a heuristic alpha-beta search method and apply it to UAlbertaBot[5], which can handle both build order planning and unit control in a small combat [15]. Gabriel *et al.* employ a neuroevolution approach to control individual unit in StarCraft, where each unit has its own network to receive input from game engine with hand-craft features, and output whether to retreat or attack in very simple scenarios [16].

As an adaptive learning method, reinforcement learning is very suitable for control problems, and there are some interesting applications of RL for micromanagement tasks. Wender *et al.* perform different RL algorithms, including Q learning and Sarsa, which have similar performances for micromanagement [17]. Shantia *et al.* use online Sarsa and neural-fitted Sarsa with a short term memory reward function to control units of the same type to attack or evade from the combat area [18].

In the last few years, the use of deep learning has dramatically improved the generalization and scalability of traditional RL algorithms. As function approximators, neural networks have been proven to be very effective and flexible in complicated environment [19]. Usunier *et al.* propose an RL method to tackle the micromanagement scenario with deep neural network [11]. They use a greedy MDP to chooses actions for units sequentially at each time step, with an zero-order optimization method. Peng *et al.* use an actor-critic method that relies on recurrent neural network (RNN) to exchange information between units [20]. Different from Usunier's and Peng's work that make use of a centralized control policy, Foerster *et al.* propose a decentralized multi-agent method in the micromanagement tasks [21]. In order to stabilize experience replay for multiple units' reinforcement learning, they condition each unit's value function on a footprint and use a multi-agent variant of importance sampling. Moreover, Foerster *et al.* use a new multi-agent actor-critic method that significantly improves average performance over centralized controllers in decentralized StatCraft combat [22].

For small scale combat in StarCraft, traditional methods have difficulty handling huge state action space, while recent methods rely on a complicated state representation and strong compute capability introduced by deep learning. In this paper, we dedicate to explore more efficient and concise state representation method to break down the complexity caused by the large state and action space, and introduce an appropriate RL algorithm to solve the problem of decision making in StarCraft combat.

III. Learning Model for Combat

A. Representation of High-Dimensional State

State representation of StarCraft combat is still an open problem and there is no universal solution. We construct a state representation with raw inputs from the game engine. This state representation method is efficient and independent of the number of units in the combat. In summary, the state representation is composed of three parts: current step state information, last step state information and last step action, as shown in Fig.2. The current step state information includes own weapon cooldown time, own unit's hitpoint, distances information of own units, distances information of enemy units and distances information of terrain. The last step state information is the same with the current step and the last step action is one-hot encoded. This mechanism is used to help the network converge and learn an appropriate policy.

All features are normalized by their maximum values. Among them, cooldown and hitpoint have 1 dimension for each. We divide the combat map into 8 sector areas on average, so that these distances information has 8 dimensions respectively. Units distance information is listed as follows:

[5]https://github.com/davechurchill/ualbertabot/wiki

own units distances are summed in the 8 directions; own units distances are maximized in the 8 directions; enemy units distances are summed in the 8 directions; enemy units distances are maximized in the 8 directions. If a unit is out of the sight range, the value is set to 0.05. Otherwise, the value is linear with d, the distance to the central unit, as demonstrated in equation (2).

$$disUnit(d) = \begin{cases} 0.05, & d > sightRange \\ 1 - 0.95(d/sightRange), & d \leq sightRange \end{cases} \tag{2}$$

In addition, terrain distance information is also computed in 8 directions and has the dimension of 8. If obstacles in the terrain is out of the sight range, the value is set to 0. Otherwise, the value is linear with the distance to the central unit, as shown in equation (3).

$$disTerrain(d) = \begin{cases} 0, & d > sightRange \\ 1 - d/sightRange, & d \leq sightRange \end{cases} \tag{3}$$

In this way, the current step state information holds the dimension of 42, as well as the last step. And the last step action has a dimension of 9, with selected action setting to 1 and the other actions setting to 0. In total, the state representation in our model is embedded to 93 dimensions.

B. Network Architecture

To train our model in the large state and action space, we use a neural network parameterized by weight $\boldsymbol{w}$ and bias $\boldsymbol{b}$ to approximate the Q value in reinforcement learning. The input of the network, $\boldsymbol{x}$, is the 93 dimensions tensor from the state representation. We has 100 neurons in the hidden layer and use a ReLU activation function for the network nonlinearity, as demonstrated in equation (4).

$$\max(0, \boldsymbol{w}^T \boldsymbol{x} + \boldsymbol{b}) \tag{4}$$

The network's output has 9 dimensions, standing for the probabilities of moving to 8 directions and attack.

C. Actions

In StarCraft combat, actions are durative, and there are approximately 24 frames per second. We use a frame skipping technique in our experiment, which makes a decision every 10 frames for each unit. And the selected actions are among the 8 move directions and an attack action for each of the living units. The attack action will make the unit stay at the current position and attack enemy units with its weapon. At present, we select the enemy with the lowest hitpoint in our weapon's fire range as the attack target. According to our experiment, these two kind of actions is enough to control our units in the small combat efficiently.

To track the problem of exploration and exploitation in reinforcement learning, the action is selected by an ϵ-greedy policy with linearly ϵ decay during training. The ϵ is initialized to 0.9 and ended to 0.01, annealing schedule with a linear smoothing window of the episode number, as shown in equation (5).

$$\epsilon = \max(0.01, 0.9 - 0.0005 * episode\_num) \tag{5}$$

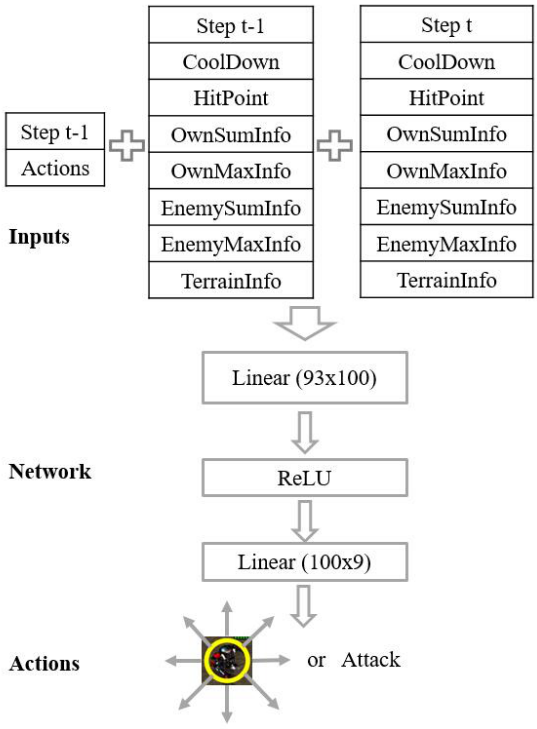

Fig. 2. Representation of the learning model for StarCraft combat. The state is represented to three parts and inputs to a network with one hidden layer. The network outputs the probabilities of moving to 8 directions and attack.

The whole model, including state representation, neural network architecture and output actions, is depicted in Fig.2.

IV. Learning Method for Combat

A. Reinforcement Learning

Reinforcement learning is a type of machine learning algorithms in which agents learn by trial and error, and determine the ideal behavior from its own experience with the environment [23]. As the most popular RL algorithm, temporal difference (TD) learning can learn directly from raw experience without a model of the environment. Besides, TD methods can update policy based on part of the sequence, without waiting for a final outcome [24]. The most widely known TD learning algorithms are Q-learning and Sarsa. Different from Q-learning's off-policy mechanism, Sarsa is an on-policy method, which means that the policy is used both for selecting actions and for updating previous Q values. The Sarsa update rule is shown in equation (6).

$$\delta_t = r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \tag{6a}$$

$$Q(s_{t+1}, a_{t+1}) = Q(s_t, a_t) + \alpha \delta_t \tag{6b}$$

δ_t is the TD error, r_{t+1} is the reward, $\gamma \in [0, 1]$ is a discount factor that determines the importance of future rewards and α is the learning rate.

B. Gradient-descent Sarsa(λ) for Multiple Units Combat

To accelerate the learning process and offset the problem of delayed reward, Sarsa with eligibility traces, Sarsa(λ), is

used. Eligibility traces are a basic mechanism of reinforcement learning to assign temporal credit, and consider a temporary history transitions such as previously observed states and taken actions [25]. This means it is not only the value for the most recently visited state-action pair but also that have been visited within a limited time before. Sarsa(λ) algorithm is one way of averaging backups made after multiple steps and λ refers to the weight of each backup. In our implementation of Sarsa(λ) for multiple units combat, we speed the learning process by sharing network parameters among our units. We learn only one network, which are used by all units. Our units can still behave differently because they receive different observations from their viewpoints and have different last step actions.

We use a neural network parameterized by vector $\boldsymbol{\theta}$ to approximate the Q value function in Sarsa(λ) and the gradient-descent learning update is shown in equation (7).

$$\delta_t = r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}; \boldsymbol{\theta}_t) - Q(s_t, a_t; \boldsymbol{\theta}_t) \tag{7a}$$

$$\boldsymbol{e}_t = \gamma\lambda \boldsymbol{e}_{t-1} + \nabla_{\boldsymbol{\theta}_t} Q(s_t, a_t; \boldsymbol{\theta}_t), \boldsymbol{e}_0 = \mathbf{0} \tag{7b}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha\delta_t \boldsymbol{e}_t \tag{7c}$$

$\boldsymbol{e}_t$ is the eligibility traces at time t. The eligibility trace of each state-action pair in the learning process is changed with the count that the state-action pair is visited.

Algorithm 1 Gradient-descent Sarsa(λ) with Neural Network

Initialize $\boldsymbol{\theta}$ arbitrarily
Repeat (for each episode):
  For each unit:
    $\boldsymbol{e}_0 = \mathbf{0}$
    Initialize s_t, a_t
    Repeat (for each step of episode):
      Take action a_t, observe r_{t+1}, next state s_{t+1}
      Choose a_{t+1} from s_{t+1} using policy derived from Q according to ϵ-greedy
      $\delta_t = r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}; \boldsymbol{\theta}_t) - Q(s_t, a_t; \boldsymbol{\theta}_t)$
      $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha\delta_t \boldsymbol{e}_t$
      $\boldsymbol{e}_{t+1} = \gamma\lambda \boldsymbol{e}_t + \nabla_{\boldsymbol{\theta}_{t+1}} Q(s_{t+1}, a_{t+1}; \boldsymbol{\theta}_{t+1})$
      $t \leftarrow t + 1$
    until s_t is terminal

To create an adaptive game AI in StarCraft combat, we use the gradient-descent Sarsa(λ) algorithms with a neural network function approximator. Gradient-descent methods are among the most widely used methods for function approximators and are particularly well suited to reinforcement learning.

C. Reward Function

Reward function has a significant impact on reinforcement learning. The goal in our combat is to destroy all of the enemy units. If the reward is based on the final result, the replay table will be extremely sparse. Moreover, getting a positive reward requires the units to explore the map to find an enemy and attack it. The delay in reward makes it difficult for the unit to learn which set of actions is responsible for what reward [26].

To tackle the problem of sparse and delayed rewards in the combat, we construct a reward function to include small intermediate rewards to speed up the convergence process. In our experiment, the attack reward is based on the difference of enemy unit's health loss and own units' health loss.

$$\begin{aligned} rewardAttack_t = (&damageAmount_t \times damageFactor \\ &-(unit\_health_{t-1} - unit\_health_t))/10 \end{aligned} \tag{8}$$

The $damageAmount$ is the damage caused by our unit's attack, and the $damageFactor$ is our units' damage information from StarCraft, e.g. 12 for Goliath in the experiment. The $unit\_health$ is the hitpoint of own units. We divided the attack reward by 10 to resize it to a more suitable range.

When a unit is destroyed, we introduce an extra negative reward, $rewardDestroyed$, and set it to -10 according to our experiment. We want to punish this behavior, in consideration that the decrease of own units' number has a bad influence on the combat result.

In order to encourage our units to work as a team and make reasonable moves, we introduce a reward for units' move actions. If these are no own units or enemy units in the move direction selected, we give this move action a small negative reward, $rewardMove$, which is set to -0.5 in the experiment.

V. Experiment

A. StarCraft Combat Scenarios

We consider combat scenarios from StarCraft as a testbed for reinforcement learning. These scenarios are challenging because the state-action space is very large and require various strategies. In the combat, each unit decides its action based on the state and action of own units and enemy units that fight against each other. As our opponent, enemy units are controlled by the built-in game AI in StarCraft, which choose to move and attack the closest using rule-based method. The combat scenario requires our units to coordinate their actions to get enemy units into their fire range, and focus fire to destroy them quickly.

TABLE I
The comparative information of both sides in the combat.

Attributes	Goliath	Zealot
Race	Terran	Protoss
Number	3	6
HitPoint	125	160
CoolDown	22	22
DamageFactor	12	16
DefenceFactor	1	1
FireRange	5	1
SightRange	8	7

In the experiment, we will control 3 Goliaths (ranged ground unit) against 6 Zealots (melee ground unit). From Table I, we can see that the enemy units have advantage on unit number, hitpoint and damagefactor. By contrast, our units' fire range is much wider. A good strategy here is keeping distance with enemies, dispersing their forces, and destroying them one by

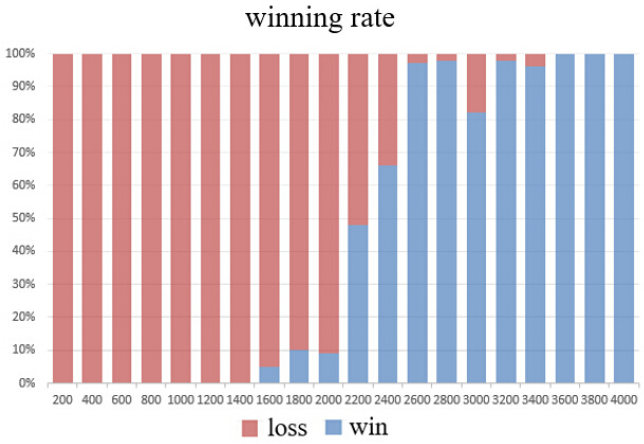

Fig. 3. The winning rate of our model from different training stages.

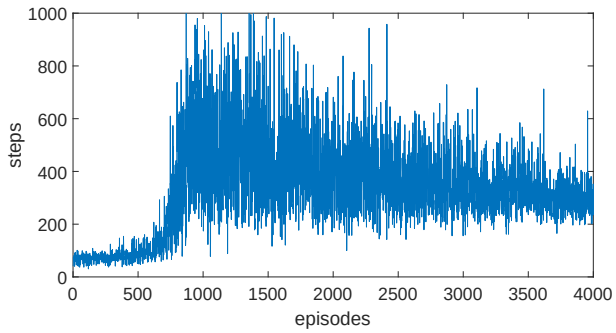

Fig. 4. The living steps during training.

one. In our experiment, the combat scenario and game AI are created with BWAPI[6].

B. Training

In order to debug and repeat the game easily, which is highly needed with RL algorithms, we use the client mode to connect our program with BWAPI. We set the discount factor γ to 0.9, the learning rate α to 0.001, and the factor in eligibility traces λ to 0.8. In order to speed the learning process, we set the gameSpeed to 0 during training, and we train the model for 4000 episodes. The experiment is applied on a computer with an Intel i7-6700 CPU and 8GB of memory.

C. Results

To evaluate the performance of the proposed method, we test our model after every 200 episodes' training for 100 combats, and depict the result in Fig.3. From the result, we can see that our units can't win any combats before 1400 episodes. With the progress of training, units start to win several combats and the winning rate has an impressive increase after 2000 episodes. After 3600 episodes' training, we can reach a winning rate of almost 100%.

[6]http://bwapi.github.io/

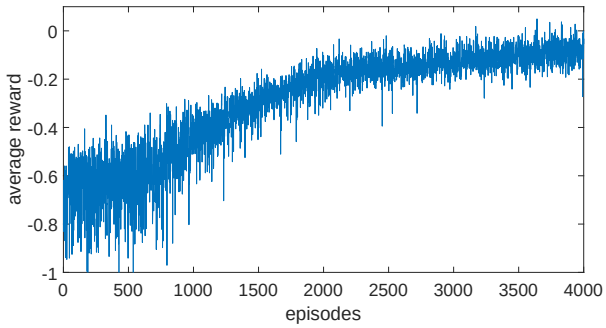

Fig. 5. The average reward during training.

We depict the average living steps of our units in an episode during training in Fig.4. From the result, it is apparent to see that the living steps curve has four stages. In the opening, the living steps is extremely low because our units have learned nothing and will be destroyed quickly. After that, our units start to realize that the health loss will cause a negative reward. As a result, they learn to run away from enemies and the living steps increase to a high level. And then, the living steps start to decrease because our units try to attack to get positive rewards, rather than just running away. In the end, our units have learned an appropriate policy to balance move and attack, and they are able to destroy enemies in almost 300 steps.

Generally speaking, a powerful game AI in the combat should defeat the enemies as soon as possible. Here we introduce the average reward, the total reward dividing the total living steps in a whole game episode. The average reward curve during training is depicted in Fig.5. We can see that the average reward has an obvious increase in the opening, grows steadily during training and stays smooth after almost 3600 episodes, when the model has converged .

At last, we use our model to play the whole combat and have a look with the learned policy. The combat can be roughly divided into 4 stages and we find our units nearly use the same policy every time. According to Fig.6, we can see that our units tend to disperse enemies into 3 parts and wipe it out in one part firstly. After that, the winning unit will move toward to other units and join the combat. Finally, our units focus fire on the remaining enemies and win the game. We choose some cooperative behaviors in the combat and draw units' move and attack directions in Fig.7. for a more intuitive understanding.

VI. Conclusion and Future Work

This paper presents several contributions, such as the gradient-descent Sarsa(λ) with neural network for multiple units combat in StarCraft, an efficient state representation method and the effective reward function for reinforcement learning. It is remarkable that the proposed method can teach our units to learn an appropriate policy and defeat the built-in AI in the combat. In the future, we will improve the method to larger scale combats, and explore more effective algorithms for multi-agent collaboration. Reward function has a great impact on reinforcement learning results, and we plan

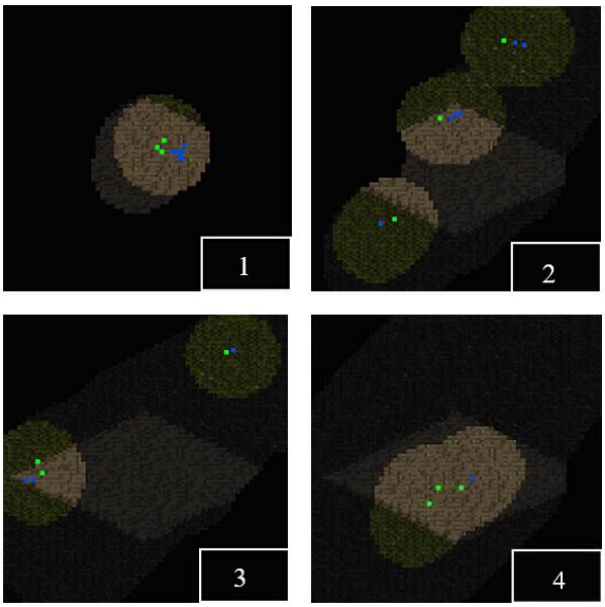

Fig. 6. The main stages during a test combat.(1) the opening stage; (2) the middle stage; (3) the later stage; (4) the final stage. The green dots stand for our units, while the blue dots stand for enemy units.

Fig. 7. The cooperative behaviors during the combat. left: support our unit in the left bottom; right: destroy the remaining enemy together. The white lines stand for the move directions and the red lines stand for the attack directions.

to further investigate and study a more appropriate reward function. In addition, we will combine tree search and deep reinforcement learning method to create a more intelligent game AI, as proved in AlphaGo.

ACKNOWLEDGMENT

The authors would like to thank Zhentao Tang and Nannan Li for the helpful comments and discussions about this work.

REFERENCES

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, and G. Ostrovski, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[2] D. Zhao, H. Wang, K. Shao, and Y. Zhu, "Deep reinforcement learning with experience replay based on SARSA," in *IEEE Symposium Series on Computational Intelligence*, 2017, pp. 1–6.

[3] D. Zhao, Z. Zhang, and Y. Dai, "Self-teaching adaptive dynamic programming for Gomoku," *Neurocomputing*, vol. 78, no. 1, pp. 23–29, 2012.

[4] K. Shao, D. Zhao, Z. Tang, and Y. Zhu, "Move prediction in Gomoku using deep learning," in *Youth Academic Annual Conference of Chinese Association of Automation*, 2016, pp. 292–297.

[5] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, d. D. G. Van, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, and M. Lanctot, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[6] M. Moravik, M. Schmid, N. Burch, V. Lisy, D. Morrill, N. Bard, T. Davis, K. Waugh, M. Johanson, and M. Bowling, "Deepstack: Expert-level artificial intelligence in heads-up no-limit poker," *Science*, vol. 356, no. 6337, pp. 508–513, 2017.

[7] N. Brown and T. Sandholm, "Safe and nested subgame solving for imperfect-information games," in *AAAI Workshop on Computer Poker and Imperfect Information Games*, 2017.

[8] S. Ontanon, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss, "A survey of real-time strategy game AI research and competition in StarCraft," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 5, no. 4, pp. 293–311, 2013.

[9] R. Lara-Cabrera, C. Cotta, and A. J. Fernndez-Leiva, "A review of computational intelligence in RTS games," in *IEEE Symposium on Foundations of Computational Intelligence*, 2013, pp. 114–121.

[10] G. Robertson and I. Watson, "A review of real-time strategy game AI," *AI Magazine*, vol. 35, no. 4, pp. 75–104, 2014.

[11] N. Usunier, G. Synnaeve, Z. Lin, and S. Chintala, "Episodic exploration for deep deterministic policies: An application to StarCraft micromanagement tasks," in *International Conference on Learning Representations*, 2017.

[12] J. Hagelback, "Hybrid pathfinding in StarCraft," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 8, no. 4, pp. 319–324, 2016.

[13] A. Uriarte and S. Ontan?n, "Kiting in RTS games using influence maps," in *AI and Interactive Digital Entertainment Conference*, 2012, pp. 31–36.

[14] G. Synnaeve and P. Bessire, "Multiscale Bayesian modeling for RTS games: An application to Starcraft AI," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 8, no. 4, pp. 338–350, 2016.

[15] D. Churchill and B. Michael, "Incorporating search algorithms into RTS game agents," in *AI and Interactive Digital Entertainment Conference*, 2012, pp. 2–7.

[16] I. Gabriel, V. Negru, and D. Zaharie, "Neuroevolution based multi-agent system for micromanagement in real-time strategy games," in *Balkan Conference in Informatics*, 2012, pp. 32–39.

[17] S. Wender and I. Watson, "Applying reinforcement learning to small scale combat in the real-time strategy game StarCraft:Broodwar," in *IEEE Conference on Computational Intelligence and Games*, 2012, pp. 402–408.

[18] A. Shantia, E. Begue, and M. Wiering, "Connectionist reinforcement learning for intelligent unit micro management in StarCraft," in *International Joint Conference on Neural Networks*, 2011, pp. 1794–1801.

[19] Y. Lecun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[20] P. Peng, Q. Yuan, Y. Wen, Y. Yang, Z. Tang, H. Long, and J. Wang, "Multiagent bidirectionally-coordinated nets for learning to play StarCraft combat games," 2017.

[21] J. Foerster, N. Nardelli, G. Farquhar, T. Afouras, P. H. S. Torr, P. Kohli, and S. Whiteson, "Stabilising experience replay for deep multi-agent reinforcement learning," in *International Conference on Machine Learning*, 2017, pp. 1146–1155.

[22] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, "Counterfactual multi-agent policy gradients," 2017.

[23] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[24] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of Artificial Intelligence Research*, vol. 4, no. 1, pp. 237–285, 1996.

[25] S. P. Singh and R. S. Sutton, "Reinforcement learning with replacing eligibility traces," *Machine Learning*, vol. 22, no. 1, pp. 123–158, 1996.

[26] A. Y. Ng, D. Harada, and S. J. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *International Conference on Machine Learning*, 1999, pp. 278–287.