Semicentralized Deep Deterministic Policy Gradient in Cooperative StarCraft Games

Dong Xie, *Student Member, IEEE*, and Xiangnan Zhong, *Member, IEEE*

***Abstract*— In this article, we propose a novel semicentralized deep deterministic policy gradient (SCDDPG) algorithm for cooperative multiagent games. Specifically, we design a two-level actor-critic structure to help the agents with interactions and cooperation in the StarCraft combat. The local actor-critic structure is established for each kind of agents with partially observable information received from the environment. Then, the global actor-critic structure is built to provide the local design an overall view of the combat based on the limited centralized information, such as the health value. These two structures work together to generate the optimal control action for each agent and to achieve better cooperation in the games. Comparing with the fully centralized methods, this design can reduce the communication burden by only sending limited information to the global level during the learning process. Furthermore, the reward functions are also designed for both local and global structures based on the agents' attributes to further improve the learning performance in the stochastic environment. The developed method has been demonstrated on several scenarios in a real-time strategy game, i.e., StarCraft. The simulation results show that the agents can effectively cooperate with their teammates and defeat the enemies in various StarCraft scenarios.**

***Index Terms*— Deep deterministic policy gradient (DDPG), multiagent system, reinforcement learning (RL), StarCraft, stochastic environment.**

I. Introduction

With the development of artificial intelligence (AI), people have been brought into a promising smart and intelligent community. Among many of the advanced techniques, reinforcement learning (RL) has been considered as one of the most important methods in the past decade [1], [2]. Recently, the development of deep neural network has endowed the conventional algorithms with new vitality [3]. For example, deep Q-network (DQN) combined the neural network techniques with the Q-learning method to solve the curse of dimensionality and accelerate the learning process [4]. Experience replay techniques and target networks also enriched DQN into a more efficient model. Furthermore, with the optimizing target network, double DQN reduced overestimate values for stability [5]. Prioritized experience replay made the batch information more useful and accelerated the agent learning process [6]. Dueling DQN splitted the Q value into two different variables in order to improve the learning efficiency in the complex environment [7].

Different from the DQN methods which make the decision based on a generated Q value, the policy gradient (PG) methods provide the opportunity to generate a recommended control action directly by a neural network [8]. An actor-critic design was developed in [9] by taking the advantages of both Q-learning and policy gradient. In this design, an actor network was built to estimate the control actions and a critic network was established to provide the corresponding value functions [10]. There are many extensions of the actor-critic design. One of the most popular methods is the deep deterministic policy gradient (DDPG) that applied experience replay and target network in the actor-critic structure to facilitate the learning process. This design focused on the continuous action space that improved the learning efficiency [11], [12]. Recently, multiagent DDPG (MDDPG) was proposed by developing cooperation and competition strategies among agents and promising results had been achieved [13], [14]. In addition, many continuous control algorithms had been put forward to improve the development of deep RL, such as asynchronous advantage actor-critic (A3C) [15], proximal policy optimization (PPO) [16], and trust region policy optimization (TRPO) [17]. In recent years, the hierarchical structure was also introduced into the RL design to improve the exploration in complex environment [18]–[20]. In particular, a hierarchical-DQN (h-DQN) was designed in [21] by establishing the hierarchical action-value functions with different temporal scales to enable the agent to learn with multistep temporal abstractions. In [22], a novel hierarchical RL structure with feudal networks (FuNs) was developed to make the agent accomplish tasks in a manager and worker architecture.

On the other hand, the development of game AI has attracted increasing attention in recent years [23]–[25], such as AlphaGo [26] and AlphaGo Zero [27] developed by the DeepMind research team. From the traditional board games to video games, AI continuously challenges and defeats the most professional human players [26]–[28]. Recently, DeepMind has introduced AI into a popular computer game as StarCraft II and proposed AlphaStar, which has defeated two professional players. AlphaStar applied the actor-critic method

Manuscript received June 19, 2019; revised February 19, 2020 and October 15, 2020; accepted November 25, 2020. This work was supported by the National Science Foundation under Grant CNS 1947418 and Grant ECCS 1947419. *(Corresponding author: Xiangnan Zhong.)*

Dong Xie is with the Department of Electrical Engineering, University of North Texas, Denton, TX 76207 USA (e-mail: dongxie@my.unt.edu).

Xiangnan Zhong is with the Department of Computer and Electrical Engineering and Computer Science, Florida Atlantic University, Boca Raton, FL 33431 USA (e-mail: xzhong@fau.edu).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TNNLS.2020.3042943.

Digital Object Identifier 10.1109/TNNLS.2020.3042943

and combined with a deep long short-term memory (LSTM) structure, an optimal policy, and a centralized value standard. This method can help the system to solve the problem with highly parallel large-scale learning in StarCraft II [29], [30]. Generally, StarCraft, which is published by Blizzard Entertainment, is considered as one of the most successful real-time strategy games in the past 20 years. The player can control different groups of factions (Protoss, Terran, and Zerg) with their units to accomplish a series of tasks and fight with enemies in various scenarios. Since StarCraft can simulate the complex and stochastic environment with different types of agents, it becomes an ideal testbed for the research and professional development of the multiagent systems [31]. Later, the release of Brood War application programming interface (BWAPI) also facilitates this study [32]. Nowadays, there are many convenient application programming interfaces including TorchCraft and PySC2 introduced in this area and can facilitate learning in the StarCraft environment [33]–[35]. In particular, SAIDA and Cherry Pi were two successful bots developed in 2018, which achieved the first two awards in StarCraft AI competition [36], [37]. Cooperating with the search algorithm, SAIDA continuously acquired and analyzed opponent information. After obtaining the ability of defense, SAIDA enabled the combat units to analyze the appropriate instants for attacking. Then, Cherry Pi applied the LSTM model to estimate the strategy and learned up to 13 different strategies based on the off-policy method. After obtaining the opponent's status and information, Cherry Pi selected the optimal strategy for the units. Moreover, StarCraft micromanagement also attracted increasing attention and many promising results have been achieved. Usunier *et al.* [38] developed a zero-order (ZO) backpropagation heuristic algorithm, which combined backpropagation with a ZO gradient estimates to help agents learn the combat strategies. A novel multiagent actor-critic method, counterfactual multiagent (COMA), was developed in [39] based on the decentralized variant design and partially observable information in the StarCraft micromanagement. A multiagent bidirectionally coordinated nets (BicNets) was established in [40] to acquire human-level coordinating strategies, such as the hit and run tactics, coordinated cover attack, and fire focus in the micromanagement scenario. In addition, the parameter sharing multiagent gradient-descent Sarsa (PS-MAGDS) algorithm [41], which combined the Sarsa method with the deep neural networks and PS-MAGDS, could defeat the enemies with fewer units in a short training time. Furthermore, QMIX [42] applied a neural network to evaluate the agent local information so that it can provide effective strategies for both centralized and decentralized policies.

Motivated by the above study, in this article, we design a semicentralized DDPG (SCDDPG) algorithm for multiagent games to cooperatively accomplish team mission. The major contributions of this article are as follows. First, this article develops a two-level actor-critic design to achieve local–global cooperation. At the local level, the generated control signal is flexible for each agent to deal with the uncertain and stochastic environment. At the global level, the agents can perform a cooperative behavior with limited centralized information, and therefore, the communication load is reduced at the same time. Second, the reward functions are designed for both levels to facilitate the learning process. In the multiagent games, each agent has a different role to guarantee the final victory. Therefore, the rewards are designed based on their own attributes to balance the attacking, position, and cooperation. Third, this method has been applied to the StarCraft computer games with various scenarios. The results have been compared with the traditional methods in the literature to show the effectiveness of the developed SCDDPG method.

The rest of this article is summarized as follows. In Section II, we introduce the background and formulate the problem considered in this article. Then, Section III provides the developed SCDDPG method. The reward functions for both local and global structures are also designed in this section. The experimental setup with StarCraft micromanagement scenarios are discussed in Section IV, and the results and analysis are provided in Section V to demonstrate the performance of the designed method. Finally, Section VI concludes our work.

II. Problem Formulation

In this section, we first formulate the StarCraft combat in a Markov decision process. Then, the conventional DDPG algorithm is discussed.

A. StarCraft Combat as a Markov Decision Process

StarCraft micromanagement process can be viewed as a multiagent Markov game. Specifically, we control a group of agents in the StarCraft game to destroy all the enemies in a certain combat environment. During the combat, each agent can interact with its neighbors and also receive local information of the environment. Therefore, the environment becomes partially observable from the viewpoint of each agent.

Developing the learning method for such a multiagent system is a challenging task. In order to build a flexible control structure for an arbitrary number of agents, we consider that each agent obtains the state s_t at time step t, including its own observation o_t of the current combat and partial teammates information c_t. The agent takes an action a_t based on the state, and then the environment returns a reward r_t and brings the agent into a new state s_{t+1}. The agents will continue this process to maximize the cumulative rewards by cooperating with the teammates and competing with the opponents.

B. DDPG

The DDPG method takes the advantages of both the actor-critic design and the DQN method to improve the exploration process. The actor in DDPG is designed to estimate the control actions, and the critic is to generate the value functions that determine the quality of the estimated actions. Specifically, the DDPG method involves four networks as the actor, the critic, and their corresponding target networks. Table I shows the input and output for each network, where θ^{μ},

TABLE I
INPUT AND OUTPUT OF EACH NETWORK IN DDPG

Structure	Network	Input	Output
Actor	Actor Network μ	s_i	$\mu(s_i\|\theta^\mu)$ as the input for the critic network Q
	Target Network μ'	s_{i+1}	$\mu'(s_{i+1}\|\theta^{\mu'})$ as the input for the critic target network Q'
Critic	Critic Network Q	s_i, a_i	$Q(s_i, a_i\|\theta^Q)$ as the predicted value for updating the critic network Q
	Target Network Q'	s_{i+1}, $\mu'(s_{i+1})$	$Q'(s_{i+1}, \mu'(s_{i+1}\|\theta^{\mu'})\|\theta^{Q'})$ as the target value for updating the critic network Q

$\theta^{\mu'}$, θ^{Q}, and $\theta^{Q'}$ are the neural network weights for the actor, actor target, critic, and critic target networks, respectively.

This method uses experience replay in the updating process. In each step, N minibatch of information (s_i, a_i, r_i, s_{i+1}) is extracted from the memory pool, where N is the batch size and i is the index of the sample in this batch. The memory pool is used to collect and store the information of the previous experience. In addition, the DDPG method introduces a random noise $\mathcal{N}_i$ that is generated by the Ornstein–Uhlenbeck process [43] to the output of the actor network $\mu(s_i|\theta^\mu)$ to facilitate the exploration in the learning process. Hence, the control signal becomes

$$a_i = \mu(s_i|\theta^\mu) + \mathcal{N}_i. \tag{1}$$

Comparing with the DQN method, DDPG uses soft target updates [11], which makes the target network taking a small change at each time step to enhance the learning stability. In this way, the four networks in the DDPG method can cooperate with each other to optimize the decision-making process.

III. SCDDPG

In this section, a novel SCDDPG method is designed to improve the control performance and reduce the communication burden at the same time. Inspired by the tree and network algorithms from the data structure, we establish a hierarchical structure with two levels of the actor-critic design. In particular, the global actor-critic design is the root node to provide the macro recommend instructions through the mapping signals c for individual agents. In order to reduce the communication burden, the global design will only receive limited information from each agent. Then, the local actor-critic design will estimate the actions by considering the recommendation along with its own local observation. In this way, each agent will take the action based on not only the local information but also the partial global information.

The developed SCDDPG method is provided in Fig. 1. Note that, each actor-critic design has its own memory pool to collect the experience of the agents and update the corresponding neural networks. The SCDDPG algorithm is expected to build a bridge between the global and the local actor-critic design that the agents can optimize their strategy based on both the local and global information.

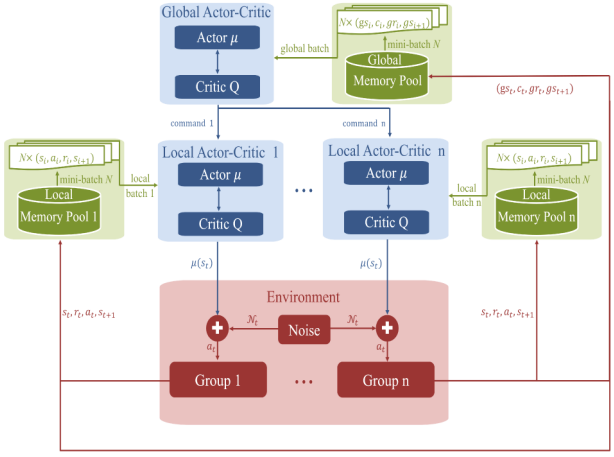

Fig. 1. Designed SCDDPG method for the multiagent system.

A. Global Actor-Critic Design

The global actor-critic design is established to estimate the mapping signals c to provide more information about the global environment to the local networks. Specifically, we design the global actor network $\mu_g(gs|\theta_g^\mu)$ with weights θ_g^μ and the global critic network $Q_g(gs, c|\theta_g^Q)$ with weights θ_g^Q. Meanwhile, we also design the target networks μ_g' and Q_g' with weights $\theta_g^{\mu'}$ and $\theta_g^{Q'}$, respectively, to help the training process. Note that, the global actor-critic design in this article can only receive partial local states from the distributed agents to reduce the communication burden. At each time step t, a random minibatch with N transition samples will be sampled from the global memory pool R_g to train the neural network weights. We define $i \in [t, t+N-1]$ as the index for the samples.

The output of the global actor network is the mapping signals c, and the input is the partial information gs received from each agent (e.g., the health values of StarCraft units). Hence, the objective function J_g for the global actor network is

$$\begin{aligned} J_g &= \mathbb{E}\left(Q_g\left(gs_i, c_i|\theta_g^Q\right)|_{\forall i\in[t,t+N-1]}\right) \\ &= \frac{1}{N}\sum_{i=t}^{t+N-1} Q_g\left(gs_i, c_i|\theta_g^Q\right), \quad i \in [t, t+N-1] \end{aligned} \tag{2}$$

where $Q_g(gs_i, c_i|\theta_g^Q)$ is the estimated global value function and N is the number of samples in the minibatch. Then, the global actor network weights θ_g^μ can be updated by

$$\frac{\partial J_g}{\partial \theta_g^\mu} \approx \frac{1}{N}\sum_{i=t}^{t+N-1} \frac{\partial Q_g\left(gs_i, c_i|\theta_g^Q\right)|_{c_i=\mu_g(gs_i)}}{\partial \mu_g\left(gs_i|\theta_g^\mu\right)} \cdot \frac{\partial \mu_g\left(gs_i|\theta_g^\mu\right)}{\partial \theta_g^\mu} \tag{3}$$

where $c_i = \mu_g(gs_i|\theta_g^\mu)$ is the output of the global actor network.

Furthermore, the input of the global critic network is defined as $[gs, c]$ and the output is the estimated global value function

$Q_g(gs, c|\theta_g^Q)$. Therefore, the loss function of the global critic network is

$$L_g = \frac{1}{N}\sum_{i=t}^{t+N-1}\left(y_{gi} - Q_g\left(gs_i, c_i|\theta_g^Q\right)\right)^2 \tag{4}$$

where y_{gi} is the approximated global target value function, which is

$$y_{gi} = gr_i + \gamma\, Q'_g\left(gs_{i+1}, \mu'_g\left(gs_{i+1}|\theta_g^{\mu'}\right)|\theta^{Q'_g}\right). \tag{5}$$

The policy gradient method is applied to train the weights in both global actor and global critic networks, and an average method is used to improve the robustness through the changes of the target networks. The weights updating law can be summarized as

$$\theta_g^{Q'} \leftarrow \tau\theta_g^Q + (1-\tau)\theta_g^{Q'} \tag{6}$$

$$\theta_g^{\mu'} \leftarrow \tau\theta_g^{\mu} + (1-\tau)\theta_g^{\mu'} \tag{7}$$

where $\tau = 0.001$ is the update parameter [11].

Note that the global actor-critic design provides the mapping signals c for each agent only based on the partial information. Comparing with the fully centralized methods, this design can reduce the communication burden by only sending a small portion of the information from each agent. This is especially important for large size learning problem where the communication is constrained. Comparing with the distributed methods, this design can guide individual agent with global information through the mapping signals, which will improve the collaboration and facilitate the learning performance.

B. Local Actor-Critic Design

The local actor-critic design is established to estimate the control actions for individual agents. In the developed method, the number of local actor-critic networks is the same as the number of agent types in one group. If a certain group only involves one type of agent, the local design will provide the distributed control signals for each agent based on their local observations.

Specifically, the goal of the local actor network is to approximate the control actions for distributed agent. The input is not only the local observation o but also the mapping signal c received from the global design

$$s = [o, c]. \tag{8}$$

We randomly select the minibatch from the local memory pool R with N transition samples. Therefore, the objective function J for the local actor network is defined as

$$\begin{aligned} J &= \mathbb{E}\left(Q\left(s_i, a_i|\theta^Q\right)|_{\forall i\in[t,t+N-1]}\right) \\ &= \frac{1}{N}\sum_{i=t}^{t+N-1} Q\left(s_i, a_i|\theta^Q\right), \quad i \in [t, t+N-1] \end{aligned} \tag{9}$$

where $Q(s_i, a_i|\theta^Q)$ is the value function estimated by the local critic network. The weights θ^μ are updated based on the chain rule as

$$\frac{\partial J}{\partial\theta^\mu} \approx \frac{1}{N}\sum_{i=t}^{t+N-1}\frac{\partial Q\left(s_i, a_i|\theta^Q\right)|_{a_i=\mu(s_i)}}{\partial\mu(s_i|\theta^\mu)}\cdot\frac{\partial\mu(s_i|\theta^\mu)}{\partial\theta^\mu} \tag{10}$$

where $\mu(s_i|\theta^\mu)$ is the output of the local actor network and the control action will be obtained as $a_i = \mu(s_i|\theta^\mu) + \mathcal{N}_i$, in which $\mathcal{N}_i$ is the exploration noise generated by the Ornstein–Uhlenbeck process.

The value function $Q(s_i, a_i|\theta^Q)$ is estimated by the local critic network with the input $[s_i, a_i]$. It is compared with the output of the local critic target network to adjust the weights and the corresponding objective function is defined as

$$L = \frac{1}{N}\sum_{i=t}^{t+N-1}\left(y_i - Q\left(s_i, a_i|\theta^Q\right)\right)^2 \tag{11}$$

where N is the sample batch size and y_i is the approximated target value function formulated by

$$y_i = r_i + \gamma\, Q'\left(s_{i+1}, \mu'\left(s_{i+1}|\theta^{\mu'}\right)|\theta^{Q'}\right) \tag{12}$$

in which $Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$ is the output of the local critic target network and $\mu'(s_{i+1}|\theta^{\mu'})$ is the output of the local actor target network.

Therefore, the local network weights updating law can be summarized as

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'} \tag{13}$$

$$\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'} \tag{14}$$

where $\tau \in [0, 1]$ is the update parameter [11].

C. Reward Function Design

We design the reward functions for both local and global designs based on the agents' attributes. This means that different types of agents are expected to learn different strategies such that they can collaborate together to improve the control performance as a group.

1) Local Reward Function: The local reward function r is designed in three parts: the hit reward r_h, the position reward r_p, and the team reward r_m

$$r = \alpha\cdot r_h + \beta\cdot r_p + \gamma\cdot r_m \tag{15}$$

where α, β, and γ are the weights for the hit, position, and team rewards, respectively. Note that the weights are designed at the beginning based on the attributes of each agent and kept the same thereafter. For example, the agents with the attribute of long-range attacking are expected to stand farther to protect themselves but still able to attack the enemies. To achieve this goal, the position reward r_p is expected to take priority in this situation. Therefore, such agents will be assigned a higher weight β comparing with α and γ. Furthermore, in the situation that the agents have the attribute of highly damaging, they are expected to always attack the enemies. Hence, such agents will be assigned a higher weight α for the hit reward r_h than the other weights.

Specifically, the hit reward r_h is designed as

$$r_h = \delta_1\cdot e_t - \delta_2\cdot u_t \tag{16}$$

where e_t is the health value difference of the nearest enemy between time step $t-1$ and t, u_t is the health value difference of the agent between time step $t-1$ and t, δ_1, and δ_2 are the adjustable weights. Note that, if the agent and the nearest

enemy are the same type, the weights δ_1 and δ_2 will be the same. If the health value of the agent attribute is less than that of the enemy, then the health value is more important. Hence, the weight δ_1 will be assigned lower than δ_2.

Then, the position reward r_p is defined as

$$r_p(d) = \begin{cases} 0, & 0 \le d \le A \\ -0.1, & A < d \le B \\ -0.5, & B < d \le C \end{cases} \tag{17}$$

where d is the distance of the nearest enemy. The areas A–C represent the agent attack range, the agent observable range, and the map boundaries, respectively.

Finally, the team reward r_m is provided as

$$r_m = m_t - o_t \tag{18}$$

where m_t and o_t are the number of the teammates and the opponents in the agent current observable range, respectively.

Note that, the weights of the rewards are designed based on the agent's attributes to perform in a certain task such that each type of the agents is able to generate their own strategies, which will enhance the collaboration between different types as a group and facilitate the learning process.

2) Global Reward Function: The global reward function is designed based on the agents' health information, number, and position of the survival teammates. The global reward includes attack reward r_a, location reward r_l, and status reward r_s, which is provided as

$$\text{gr} = \kappa \cdot r_a + \zeta \cdot r_l + \xi \cdot r_s \tag{19}$$

where κ, ζ, and ξ are the weights for the attack, location, and status rewards, respectively. For example, if the number of the survival teammates is less than that of the survival enemies, the location reward r_l and the status reward r_s are more important than the attack reward r_a. Therefore, ζ and ξ will be assigned higher than κ.

Specifically, the attack reward r_a is defined as

$$r_a = \rho_1 \cdot \sum_{j=1}^{m} e_t^j - \rho_2 \cdot \sum_{j=1}^{n} u_t^j \tag{20}$$

where $\sum_{j=1}^{m} e_t^j$ is the health value difference of all the survival enemies between time step $t-1$ and t, $\sum_{j=1}^{n} u_t^j$ is the health value difference of all the survival agents between time step $t-1$ and t, m and n are the numbers of survival enemies and agents, respectively, and ρ_1 and ρ_2 are the health value weights for enemies and agents, respectively, in which ρ_1 and ρ_2 are determined by the agent attributes and the number of survival teammates and enemies.

Then, the location reward r_l is described as

$$r_l = \frac{1}{n} \sum_{j=1}^{n} r_p^j \tag{21}$$

$$r_p^j = \begin{cases} 0, & 0 \le d^j \le A \\ -0.1, & A < d^j \le B \\ -0.5, & B < d^j \le C \end{cases} \tag{22}$$

TABLE II
ATTRIBUTES OF SIX UNITS IN STARCRAFT COMPUTER GAME

Attributes	Vulture	Zergling	Dragoon	Zealot	Marine	Wraith
Hitpoints	80	35	100	100	40	120
Range	5	1	4	1	4	5
Sight	8	5	8	7	7	7
Cooldown	1.26	0.336	1.26	0.924	0.63	0.924
Damage	20	5	20	8	6	20

where d^j is the distance between the jth agent and its nearest enemy and $1/n \sum_{j=1}^{n} r_p^j$ is the average of the local position reward for all the survival agents. Areas A, B, and C represent the jth agent attack range, observable range, and map boundaries, respectively.

Moreover, the status reward r_s is designed as

$$r_s = \tau_1 \cdot n - \tau_2 \cdot m \tag{23}$$

where n and m are the numbers of survival agents and enemies, respectively, and τ_1 and τ_2 are the weights that are designed based on the total number of the agents and enemies.

In this method, the weights adjustments are involved in both local and global reward function design. They are provided based on the attributes of the agents. All the rewards are normalized in $[0, 1]$ during the learning process. The developed SCDDPG method is summarized in Algorithm 1. Note that the random process $\mathcal{N}$ in line 17 is generated by the Ornstein–Uhlenbeck process [43] as the action noise to improve the exploration during the learning process. This exploration noise will be multiplied by an exploration rate, which is decreased every time step until it reaches the threshold and stays thereafter. This design is expected to balance the exploration and exploitation for the action selection.

IV. EXPERIMENTAL SETUP AND DESIGN

The StarCraft computer game is considered in this section to demonstrate the effectiveness of our developed SCDDPG method. The experimental scenarios and the parameters setup are discussed. The experiment design refers to the literature [38], [40]. We configure the StarCraft server with BWAPI and use TorchCraft with Python to train the agents in the StarCraft games [32], [33].

A. StarCraft Scenario Design

In this article, we apply the developed SCDDPG method on five StarCraft scenarios involving six unit types as Dragoon, Marine, Vulture, Wraith, Zergling, and Zealot, from all three factions (Protoss, Terran, and Zerg). We also compare our results with other RL algorithms in the literature to further validate the developed method. The attributes of the StarCraft units involved in the experiment are provided in Table II.

Since StarCraft is a real-time strategy game, the operational delay is inevitable. Therefore, we set the frameskip as 10 during the experiment [38], [40]. This means that every action only lasts ten frames in the game.

Algorithm 1 SCDDPG Algorithm

Initialize global critic network $Q_g(gs, c|\theta_g^Q)$ with weights θ_g^Q and global actor network $\mu_g(gs|\theta_g^\mu)$ with weights θ_g^μ.
Initialize global target network Q'_g with weights $\theta_g^{Q'}$ and μ'_g with weights $\theta_g^{\mu'}$
Initialize replay buffer R_g with previous model memory
for agent type j = 1, K **do**
  Initialize local critic network $Q_j(s, a|\theta_j^Q)$ with weights θ_j^Q and local actor network $\mu_j(s|\theta_j^\mu)$ with weights θ_j^μ
  Initialize local target network Q'_j with weights $\theta_j^{Q'}$ and μ'_j with weights $\theta_j^{\mu'}$
  Initialize replay buffer R_j with previous model memory
end for
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation agent state o_1
  Select command c_1 from global actor-critic
  Obtain new state $s_1 = [o_1, c_1]$
  for t = 1, T **do**
    Select command c_{t+1} from global actor
    for agent type j = 1, K **do**
      Select action $a_t = \mu_j(s_t|\theta_j^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
      Execute action a_t at and observe reward r_t and observe new agent state o_{t+1}
      Obtain new state $s_{t+1} = [o_{t+1}, c_{t+1}]$
      Store transition (s_t, a_t, r_t, s_{t+1}) in R_j
      Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R_j
      Update local critic network with equation (11)(12)
      Update local actor network with equation (9)
      Update local target networks with equation (13)(14)
    end for
    Observe global reward gr_t and extract new global state gs_{t+1}
    Store transition $(gs_t, c_t, gr_t, gs_{t+1})$ in R_g
    Sample a random minibatch of N transitions $(gs_i, c_i, gr_i, gs_{i+1})$ from R_g
    Update global critic network with equation (4)(5)
    Update global actor network with equation (2)
    Update global target networks with equation (6)(7)
  end for
end for

We consider the StarCraft units as the agents in the RL design. Then, based on the attributes, we group the units in the following five scenarios.

1) *Two Dragoons and Three Zealots Versus Two Dragoons and Three Zealots (2D3Zv2D3Z):* Each group has two types of units and the number of units in each type is the same. Therefore, two local actor-critics and one global actor-critic are developed in the SCDDPG method. One local actor-critic structure is design for two Dragoons, and the other one serves for three Zealots. The global actor-critic would give a set of mapping signals for each individual unit.
2) *Three Vultures and Ten Zerglings versus Three Vultures and 12 Zerglings (3V10Zv3V12Z):* Similar to the first scenario, each group involves two types of units. However, the enemy has two more Zerglings than our group.
3) *Five Marines Versus Five Marines (5Mv5M):* Five Marines join the combat in each group. The SCDDPG method is designed with one local and one global actor-critic to control one group of Marines.
4) *15 Marines Versus 16 Marines (15Mv16M):* 15 Marines are in our group, and the enemy includes 16 Marines.
5) *15 Wraiths Versus 17 Wraiths (15Wv17W):* 17 Wraiths are controlled by the opponent, while our group has 15 Wraiths.

Therefore, the combat involves two groups of units. Some of the groups consist of different types of units. One group is controlled by the developed SCDDPG method, whereas the other one is led by the StarCraft game heuristic AI. The goal of both groups is to eliminate all the opponents.

Note that both groups involve similar units' structures, and in some scenarios, the heuristic AI-controlled group also takes the advantage of the number of units, which makes the task more challenging. Therefore, the SCDDPG method will design different reward functions for different types of units based on their attributes such that they can learn different strategies and collaborate with each other to guarantee the final victory as a group.

B. Training With SCDDPG Method

The developed SCDDPG method is applied for the StarCraft games. The learning rates of the actor and critic networks are set to be 1e−6 and 1e−5, respectively. The discount factor is defined as 0.99. The exploration rate is set to be 1 at the beginning and is decreased by 6e−6 every time step until it reaches 0.4 and stays thereafter. Furthermore, the size of the memory pool is defined as 1e7 and the batch size is 32.

The local actor-critic design has four inputs: the local state, the action and the reward function at time step t, and the local state at time step $t + 1$. The local state vector for each unit includes 28 variables, three of which is the global output c. The remaining 25 variables are the unit information obtained from BWAPI and the map. The detailed description is shown next. Note that all the variables are normalized within [0, 1].

1) Health: Six variables are included: the unit current health value, the unit health value level, the enemy current health value, the enemy health value level, the unit survival status, and the enemy survival status.

2) Number: Four variables are included: the number of the survival units, the number of the survival enemies, and the number of the teammates and the enemies within the unit observable range.

3) Attack: Six variables are the unit attack cooldown value, the enemy attack cooldown value, the unit action, the enemy action, the unit under-attacking information, and the enemy under-attacking information.

4) Position: Nine variables are the unit map location (x, y), the enemy map location (x, y), the nearest enemy distance coordinates (dx, dy, d), where dx, dy, and d are the horizontal, the vertical, and the minimum distance, respectively,

and the enemy attack target map location (x, y). If there is no enemy within the observation, the variable is set as the distance between the agent and the map boundary.

5) Command: Three variables are included: the global mapping signal c received from the global actor-critic network.

Furthermore, the global actor-critic design also has four inputs: the global state, the command, and the reward function at time step t, and the global state at time step $t+1$. Note that, only the 6-D health information is considered as the global state. For example, in the 2D3Zv2D3Z scenario, the total number of the unit is five, and therefore, the dimension of the global state is $6 \times 5 = 30$. This means that the distributed unit can send the health information only to the global level. Comparing with the centralized method, this design can reduce the communication burden and also provide global information to facilitate the collaboration among units. Both the global and the local neural networks are established with two hidden layers and 150 neurons in each hidden layer. A rectifying linear unit is established as the activation function for the neural network structure to help with the learning process. The Adam optimizer is also applied to update the neural network weights.

Moreover, the output of the local actor-critic design is a 3-D action, which is an instruction with a position coordinates (z, x, y), where the instruction z will decide whether to move or to attack and the position coordinates (x, y) decide the moving direction. All these three variables are normalized within $[-1, 1]$. Hence, the unit will attack the nearest enemy if the instruction $z > 0$. If the nearest enemy is not within the range of unit fire, the unit will move to the nearest enemy with the position coordinates (x, y). If the instruction $z < 0$, the unit will take the move action with a certain direction $\tan^{-1}(x/y)$ that is calculated by the position coordinates (x, y). For example, if the output is $[-0.9, -0.25, 0.5]$ with the instruction $z = -0.9 < 0$, then the unit will move with the direction of $\tan^{-1}(-0.25/0.5) = -30°$. In addition, the output of the global actor-critic design is the mapping signal c that is a 3-D vector to provide the global health information for distributed units.

The memory pools are designed for both the global and the local structures. Specifically, the local memory pool saves the current local state, action, reward, and next state of the unit as one batch for memory replay training and the global memory pool stores the global state, action, reward, and next state. During the training process, at each time step, the stored experience will be randomly selected from each memory pool based on the batch size to update the corresponding neural network. Each type of units will have its own memory pool. Therefore, each type of unit has its own actor-critic structure with the memory pool corresponding with its own experience. Meanwhile, all types of units share one global actor-critic design with limited information received from the distributed units.

V. Experiment Results and Analysis

A. Experimental Evaluation Index

Three evaluation indices are provided to demonstrate and evaluate the effectiveness of the developed SCDDPG method, which are the win rate, the normalized reward, and the survival rate. These evaluation indices are calculated at each episode as the average for the last 100 episodes. The details of the evaluation indices are provided as follows.

First, the win rate is the percentage of the winning times, which can be defined as

$$\text{win rate: } W = \frac{v}{100} \tag{24}$$

where v is the number of victories within 100 episodes.

Then, the normalized reward is the average of the cumulative rewards, which is

$$\text{normalized reward: } N = \frac{\sum_{k=1}^{100} \sum_{j=1}^{n} r_{kj}}{n \cdot 100} \tag{25}$$

where k is the index of episode, j is the index of unit, n is the number of units, and r_{kj} is the final reward for unit j at episode k.

Finally, the survival rate is the average rate of the survival units within 100 episodes, which is

$$\text{survival rate: } S = \frac{\sum_{k=1}^{100} l_k}{n \cdot 100} \tag{26}$$

where l_k is the number of the survival units in episode k. Note that, all the evaluation indices are normalized within $[0, 1]$.

B. Scenario Result Analysis

We demonstrate and discuss the experimental results based on the three evaluation indices (24)–(26). In addition, to further validate the developed method, we also compare the results with the literature [38], [40] under the same experimental setup.

1) Scenario One: This scenario considers the situation of 3V10Zv3V12Z, which is shown in Fig. 2. The SCDDPG method is applied, and 1000 episodes (games) have been conducted. In particular, for each episode, 13 units have been controlled by the SCDDPG method, including three Vultures and ten Zerglings. The enemy group has the same unit-type structure and is under controlled by the StatCraft build-in game heuristic AI. If our group eliminates all the enemies, we consider that the group wins the game in this episode; otherwise, it loses.

The results are shown in Fig. 3, including the trajectories of the win rate, the normalized reward, and the survival rate. Specifically, from the trajectory of the win rate, in the first 100 episodes, the SCDDPG-controlled group has not obtained one win episode, and the enemy wins all games. However, in the following 300 episodes, the win rate of the controlled team increased rapidly from 0 to 0.82. After 600 episodes of training, the win rate is over 0.9 and reaches a stable period, which means that the group under the controlled by SCDDPG method has over 90% chance to win the game.

The rapid growth of the normalized reward comes in the second 100 episodes, and the normalized reward is from 0.27 to 0.64. At the same time, the corresponding win rate continuously increases to 0.25, which means that the SCDDPG team steadily turns the table from being in the passive position

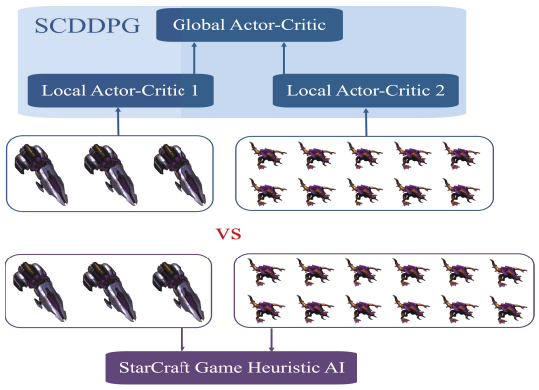

Fig. 2. Structure of scenario 3V10Zv3V12Z.

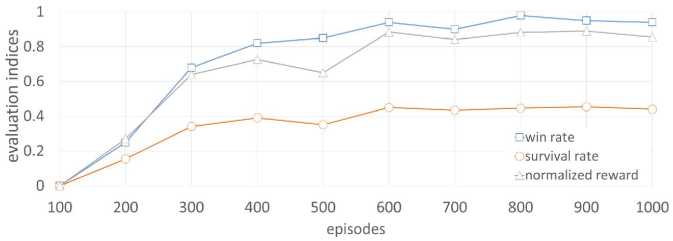

Fig. 3. Evaluation indices for scenario 3V10Zv3V12Z including win rate, normalized reward, and survival rate.

to actively participate in the battle. In the following episodes, normalized reward increases smoothly. After 600 episodes, the normalized reward is stable at around 0.9. With the help of the local and the global reward function, the intelligent multiagent system can collaborate with each other to achieve the final victory as a group.

Furthermore, the survival rate is 0 in the first 100 episodes because the SCDDPG team loses all games, and there is no unit in the team survives. After exploiting and training of 300 episodes, the survival rate in each battle increases from 0 in the first 100 rounds to more than 0.38 in the fourth 100 rounds. After training 600 episodes, since the win rate and the normalized reward become stable, the survival rate keeps above 0.4, which means that after destroying all enemy team members, our team still has five units alive. In other words, although we lose eight units, we eliminate 15 enemies and win the game.

2) Scenario Two: In this scenario, we consider the scenario with 2D3Zv2D3Z. Without loss of generality, the experiment is conducted with 100 independent runs, where each run includes 1000 episodes. In order to demonstrate the effectiveness of the proposed method, we also compare the results of the SCDDPG method with the DDPG design. The parameters of both methods are selected as the same, such as the neural network learning rate, discount factor, exploration rate, and batch size, among others.

The battle scene of this scenario is shown in Fig. 4. At the beginning of each episode, the SCDDPG method controls five red units, including two Dragoons and three Zealots, while the enemy has the same five units with color blue. Fig. 5 shows the win rate comparisons of the SCDDPG and the DDPG methods. In the first 100 episodes, the SCDDPG-controlled group only

Fig. 4. StarCraft battle scene of scenario 2D3Zv2D3Z. The units are surrounded by fog. The map settings are based on [38].

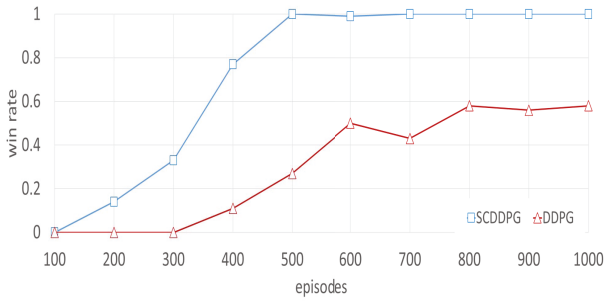

Fig. 5. Win rate comparisons of SCDDPG and DDPG methods for scenario 2D3Zv2D3Z.

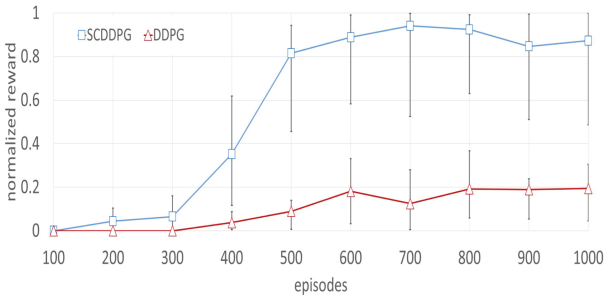

Fig. 6. Normalized reward comparisons of SCDDPG and DDPG methods for scenario 2D3Zv2D3Z.

wins one episode, while the enemy group wins 99 games. After that, the SCDDPG group continuously optimizes its strategy. In the next 400 rounds, the win rate of the SCDDPG method shows and explosive growth and increases from 0 to 1. After 500 episodes, the SCDDPG group keeps to optimize the decision and the enemies are unable to win the game. On the other hand, the DDPG method cannot achieve one victory in the first 300 episodes, and after that, the win rate increases slowly. At the end of the training process, the DDPG method can reach stable a win rate at 0.6.

The comparisons of the average performance of 100 independent runs are shown in Figs. 6 and 7 in terms of the normalized reward and the survival rate. Specifically, the average normalized reward is presented in Fig. 6. The increasing trend of the normalized reward is consistent with the win rates. For the SCDDPG method, the normalized reward is about

TABLE III
WIN RATES OF SIX ALGORITHMS IN FIVE STARCRAFT SCENARIOS [38], [40]

Scenarios	**SCDDPG**	DDPG	DQN	PG	ZO	BicNet
2D3Zv2D3Z	**1.00**	0.60	0.61	0.69	0.90	–
3V10Zv3V12Z	**0.95**	0.45	–	–	–	–
5Mv5M	**0.94**	0.56	0.99	0.92	1.00	0.92
15Mv16M	**0.93**	0.89	0.13	0.19	0.79	0.71
15Wv17W	**0.56**	0.27	0.16	0.14	0.49	0.53

SCDDPG: semi-centralized deep deterministic policy gradient DDPG: deep deterministic policy gradient DQN: deep Q-network PG: policy gradient ZO: zero-order BicNet: multi-agent bidirectionally-coordinated nets

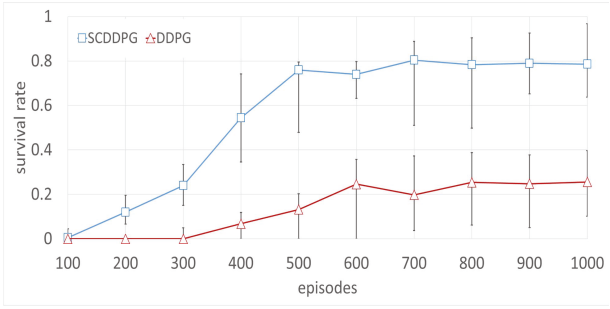

Fig. 7. Survival rate comparisons of SCDDPG and DDPG methods for scenario 2D3Zv2D3Z.

0.1 after the first 100 episodes training. At first, the reward difference between the SCDDPG and the DDPG methods is quite small. However, after 400 episodes, the normalized reward of SCDDPG exceeds 0.3 and the normalized reward of DDPG still stays in 0.05. Then, the trajectory of the SCDDPG normalized reward has shown explosive growth and achieves 0.8 within 500 episodes of training. Then, the strategy of SCDDPG is kept training in the following 400 episodes and the normalized reward maintains around 1.0. However, the normalized reward of DDPG increases slowly comparing with the SCDDPG method in the training process and is lower than 0.2 until the end. Furthermore, Fig. 7 shows the average survival rate of 100 runs. In the first 100 episodes, the survival rates of SCDDPG and DDPG are 0.005 and 0, respectively. After that, the survival rate of SCDDPG shows a rise and increases to 0.77 after 500 episodes of training. Then, during the training episodes between 600 and 700, the survival rate breaks 0.8, which means that we lost only one unit but defeated five enemies. Thus, the SCDDPG method can achieve a convincing performance in this scenario. In the DDPG method, the survival rate trajectory is similar to the DDPG normalized reward, and the DDPG method can only save about one or two units at the end of each game after 1000 episodes training.

C. Experimental Comparison of Algorithms

To further demonstrate the SCDDPG method, we compare our results with the other five methods: DDPG, DQN, PG, ZO, and BicNet. Note that, we take the experiment on SCDDPG and DDPG, and the other results are from the literature [38], [40]. For both methods, we train the StarCraft units for 1000 episodes and fix the neural network weights for testing. The comparisons are shown in Table III.

In scenario 2D3Zv2D3Z, SCDDPG obtains a 100% win rate, which is 10% higher than the second-best performance algorithm ZO. In 3V10Zv3V12Z, although DQN, PG, ZO, and BicNet have no corresponding experiments, the SCDDPG method obtains a 0.95 win rate, which is much better than DDPG. Moreover, the SCDDPG method has competitive results in the 5Mv5M scenario. In scenario 15Mv16M, SCDDPG significantly improves the win rate compared with other methods. In the last scenario 15Wv17W, all algorithms cannot guarantee to acquire a higher victory rate because of the difficult setup of the game. However, our method can still obtain a better performance. Therefore, the results show that our designed SCDDPG method has better control performance, especially for powerful opponents.

VI. CONCLUSION AND DISCUSSION

This article proposed an SCDDPG method for multiagent systems. The method involved two-level structures for the global and the local actor-critic design. The reward functions were also developed for different types of agents based on their attributes to improve collaboration during the learning process. The method had been implemented on the StarCraft micromanagement and compared with other methods to demonstrate the effectiveness. The results showed that our developed method can optimize the game and improve collaboration among different types of StarCraft units.

In this article, we focused on the games between two groups and the results had been compared with the literature. In the future, we would like to consider the games involving multiple groups, which requires the intelligent group to decide which groups it should collaborate with first to eliminate the other groups. They have to communicate with the target groups on an agreement for collaboration.

REFERENCES

[1] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *J. Artif. Intell. Res.*, vol. 4, no. 1, pp. 237–285, Jan. 1996.
[2] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Netw.*, vol. 61, pp. 85–117, Jan. 2015.
[3] Y. Li, "Deep reinforcement learning: An overview," 2017, *arXiv:1701.07274*. [Online]. Available: http://arxiv.org/abs/1701.07274
[4] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[5] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proc. AAAI*, Phoenix, AZ, USA, vol. 2, 2016, p. 5.

[6] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," 2015, *arXiv:1511.05952*. [Online]. Available: http://arxiv.org/abs/1511.05952

[7] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, "Dueling network architectures for deep reinforcement learning," 2015, *arXiv:1511.06581*. [Online]. Available: http://arxiv.org/abs/1511.06581

[8] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Proc. Adv. Neural Inf. Process. Syst.*, 2000, pp. 1057–1063.

[9] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *Proc. Adv. Neural Inf. Process. Syst.*, 2000, pp. 1008–1014.

[10] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.

[11] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," in *Proc. Int. Conf. Learn. Represent.*, 2016, pp. 1–14.

[12] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1329–1338.

[13] J. K. Gupta, M. Egorov, and M. Kochenderfer, "Cooperative multi-agent control using deep reinforcement learning," in *Proc. Int. Conf. Auton. Agents Multiagent Syst.* Cham, Switzerland: Springer, 2017, pp. 66–83.

[14] R. Lowe, Y. Wu, A. Tamar, J. Harb, O. P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 6379–6390.

[15] V. Mnih *et al.*, "Asynchronous methods for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1928–1937.

[16] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*. [Online]. Available: http://arxiv.org/abs/1707.06347

[17] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 1889–1897.

[18] A. G. Barto and S. Mahadevan, "Recent advances in hierarchical reinforcement learning," *Discrete Event Dyn. Syst.*, vol. 13, nos. 1–2, pp. 41–77, 2003.

[19] O. Nachum, S. S. Gu, H. Lee, and S. Levine, "Data-efficient hierarchical reinforcement learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 3303–3313.

[20] C. Florensa, Y. Duan, and P. Abbeel, "Stochastic neural networks for hierarchical reinforcement learning," 2017, *arXiv:1704.03012*. [Online]. Available: http://arxiv.org/abs/1704.03012

[21] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum, "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 3675–3683.

[22] A. S. Vezhnevets *et al.*, "Feudal networks for hierarchical reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, vol. 70, 2017, pp. 3540–3549.

[23] M. Buro, "Real-time strategy games: A new AI research challenge," in *Proc. Int. Joint Conf. Artif. Intell.*, 2003, pp. 1534–1535.

[24] R. Lara-Cabrera, C. Cotta, and A. J. Fernandez-Leiva, "A review of computational intelligence in RTS games," in *Proc. IEEE Symp. Found. Comput. Intell. (FOCI)*, Apr. 2013, pp. 114–121.

[25] G. N. Yannakakis and J. Togelius, *Artificial Intelligence and Games*. Springer, 2018.

[26] D. Silver *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, p. 484, 2016.

[27] D. Silver *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354, 2017.

[28] V. Mnih *et al.*, "Playing Atari with deep reinforcement learning," 2013, *arXiv:1312.5602*. [Online]. Available: http://arxiv.org/abs/1312.5602

[29] K. Arulkumaran, A. Cully, and J. Togelius, "AlphaStar: An evolutionary computation perspective," 2019, *arXiv:1902.01724*. [Online]. Available: http://arxiv.org/abs/1902.01724

[30] O. Vinyals *et al.*, "Grandmaster level in StarCraft II using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, Nov. 2019.

[31] M. Buro and D. Churchill, "Real-time strategy game competitions," *AI Mag.*, vol. 33, no. 3, p. 106, Sep. 2012.

[32] A. Heinermann. (2015). *BWAPI: Brood War API, an API for Interacting With starcraft: Broodwar (1.16.1)*. [Online]. Available: https://bwapi.github.io

[33] G. Synnaeve *et al.*, "TorchCraft: A library for machine learning research on real-time strategy games," 2016, *arXiv:1611.00625*. [Online]. Available: http://arxiv.org/abs/1611.00625

[34] 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 Trans. Comput. Intell. AI Games*, vol. 5, no. 4, pp. 293–311, Dec. 2013.

[35] D. Churchill *et al.*, "Starcraft bots and competitions," in *Encyclopedia of Computer Graphics and Games*. 2016, pp. 1–18.

[36] S. Xu, H. Kuang, Z. Zhuang, R. Hu, Y. Liu, and H. Sun, "Macro action selection with deep reinforcement learning in StarCraft," 2018, *arXiv:1812.00336*. [Online]. Available: http://arxiv.org/abs/1812.00336

[37] D. Churchill, M. Buro, and R. Kelly, "Robust continuous build-order optimization in StarCraft," in *Proc. IEEE Conf. Games (CoG)*, Aug. 2019, pp. 1–8.

[38] N. Usunier, G. Synnaeve, Z. Lin, and S. Chintala, "Episodic exploration for deep deterministic policies: An application to StarCraft micromanagement tasks," 2016, *arXiv:1609.02993*. [Online]. Available: http://arxiv.org/abs/1609.02993

[39] J. N. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, "Counterfactual multi-agent policy gradients," in *Proc. 32nd AAAI Conf. Artif. Intell.*, 2018.

[40] P. Peng *et al.*, "Multiagent bidirectionally-coordinated nets: Emergence of human-level coordination in learning to play StarCraft combat games," 2017, *arXiv:1703.10069*. [Online]. Available: http://arxiv.org/abs/1703.10069

[41] K. Shao, Y. Zhu, and D. Zhao, "StarCraft micromanagement with reinforcement learning and curriculum transfer learning," *IEEE Trans. Emerg. Topics Comput. Intell.*, vol. 3, no. 1, pp. 73–84, Feb. 2019.

[42] T. Rashid, M. Samvelyan, C. S. Witt, G. Farquhar, J. Foerster, and S. Whiteson, "QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 4292–4301.

[43] G. E. Uhlenbeck and L. S. Ornstein, "On the theory of the Brownian motion," *Phys. Rev.*, vol. 36, p. 823, Sep. 1930.

Dong Xie (Student Member, IEEE) received the B.S. degree in information and computing science from Shandong Agricultural University, Tai'an, China, in 2015, and the M.S. degree in applied mathematics from George Washington University, Washington, DC, USA, in 2017. He is currently pursuing the Ph.D. degree with the Department of Electrical Engineering, University of North Texas, Denton, TX, USA.

He was a Graduate Research Assistant at Johns Hopkins University, Baltimore, MD, USA, and Fujitsu, Beijing, China, before. His research interests include reinforcement learning, computer vision, and natural language processing.

Xiangnan Zhong (Member, IEEE) received the B.S. and M.S. degrees in automation, and control theory and control engineering from Northeastern University, Shenyang, China, in 2010 and 2012, respectively, and the Ph.D. degree with the Department of Electrical, Computer, and Biomedical Engineering, University of Rhode Island, Kingston, RI, USA, in 2017.

From 2017 to 2019, she was an Assistant Professor with the Department of Electrical Engineering, University of North Texas, Denton, TX, USA. She is currently an Assistant Professor with the Department of Computer and Electrical Engineering and Computer Science, Florida Atlantic University, Boca Raton, FL, USA. Her current research interests include computational intelligence, reinforcement learning, cyber-physical systems, networked control systems, neural network, and optimal control.

Dr. Zhong was a recipient of the International Neural Network Society Doctoral Dissertation Award in 2019. She has been actively involved in numerous conference and workshop organization committees in society.