Hindawi
Wireless Communications and Mobile Computing
Volume 2022, Article ID 5340517, 11 pages
https://doi.org/10.1155/2022/5340517

Research Article

Performance Evaluation of Multiagent Reinforcement Learning Based Training Methods for Swarm Fighting

Huanli Gao, Yahui Cai, He Cai, Haolin Lu, and Jiahui Lu

School of Automation Science and Engineering, South China University of Technology, Guangzhou 510641, China

Correspondence should be addressed to He Cai; caihe@scut.edu.cn

Received 13 August 2022; Accepted 24 September 2022; Published 11 October 2022

Academic Editor: Jun Ye

In this paper, we conducted a performance evaluation of two multiagent reinforcement learning based training methods for swarm fighting, namely, the multiagent reinforcement learning (MARL) training method, and the combined multiagent reinforcement learning and behavior cloning (MARL-BC) training method. The behavior cloning expert is taken from some well-trained model in the final steady phase by the MARL training method. From the perspective of winning rate, the performances of these two different training methods can be divided into three phases. In the first phase, learning progresses slowly for both these two training methods. As the model trained by the MARL training method grows stronger, the experience of the behavior cloning expert gradually becomes useful, and the second phase kicks off where the MARL-BC training method takes obvious advantage. Surprisingly, the advantage of the MARL-BC training method will disappear as the learning progress goes on because in this final phase the expert of the behavior cloning training method can no longer offer the right strategy in presence of the ever changing environment and opponent.

1. Introduction

The core mechanism of reinforcement learning (RL) is feedback. In the process of RL, agents perceive the environment, take actions based on observation, and receive feedback from the environment to adjust their actions. By extensive trial and error, agents tend to exhibit the desired behavior expected by the trainer. RL has also inspired many other learning methods, such as Q learning [1], Neuro-Dynamic Programming [2], Policy Gradient Learning [3], and so on. These novel RL related learning methods have drastically improved the ability of RL from the perspective of perception and expression, thus facilitating the application of RL in many scenarios, such as robotics, computer vision, health, transportation, finance, games, autonomous driving, natural language processing, and other aspects [4–13].

Multiagent reinforcement learning (MARL) is an important research direction of RL because the problem that can be solved by a single agent is very limited. In the scenario of swarm fighting, multiagent systems should make timely judgment according to the change of the environment and take action in the next move, such as attacking, avoiding, encircling, and cheating. In order the win the fighting, the multiagent system need to learn how to collaborate in the process of reinforcement learning, and strive to maximize the benefits in the changing environment. For example, for unmanned aerial vehicle air combat, the environment is evolving with high dynamic subject to complex conditions, such as intermittent signal interference and dense fire threat. Thus, it would be impossible to obtain a feasible fighting strategy by a single and centralized learning method. Instead, distributed and collaborative decision-making by multiple agents would be necessary. However, the behaviors of multiagent systems are essentially unstable Markov decision-making process, which makes it extremely difficult, if possible, to obtain any affirmatively effective solution by rule-based methods. On the contrary, it is the very property of RL to effectively deal with complex and dynamic environment. Recently, the MARL method proves to be a promising way to solve the swarm fighting problem [14]. The players on both sides in the swarm fighting constantly interact with the environment, learn the reward value of each behavior through RL, and combine these reward values to find the optimal strategy. The multiagent deep deterministic policy gradient was proposed in [15], which established a framework featuring centralized training and decentralized execution. Reference [16] proposed the Deep Reinforcement Opponent Network algorithm with two neural networks, where one neural network is designed to evaluate its own Q value, while the other neural network is designed to learn policy representations for other agents. In this way, the intentions of other agents can be understood which helps to improve algorithm performance. In [17], a novel cooperative multiagent reinforcement learning method was conceived for both discrete and continuous action spaces, called FACtored Multiagent Centralised policy gradients (FACMCA). Like the Multiagent Deep Deterministic Policy Gradient method, FACMCA also uses depth determination policy gradients to learn policies. However, FACMCA learns a centralized but decomposed critic that combines the application of each agent into a joint action-value function via a nonlinear monotonic function. The authors of [18] utilized additional state information in MARL and proposed a new actor-critic framework, namely value-decomposition actor-critic (VDAC). This framework achieves a reasonable balance between training efficiency and algorithm performance. As the number of agents increases, the input of centralized reviewers under the centralized training distribution execution framework increases linearly, increasing the difficulty of training. To address this issue, [19] introduced an attention mechanism to solve the issues of dataset and training. Efficient communication is crucial for effective cooperation of agents. [20] explored the communication among agents and proposed the Multiagent Bidirectionally-Coordinated Network. A communication channel through a two-way recurrent neural network was built, and the agents could store local information while communicating continuously. RL may encounter the problem of sparse rewards during training, i.e., the agent does not get any reward after many decision-making steps, which makes the learning process of the agent difficult [21]. In such cases, the researchers thought of using behavior cloning to speed up the learning progress, i.e., the agent imitates the behavior of the expert, making its own behavior close to the expert. Behavior cloning was invoked in [22] for autonomous driving. By collecting action-observation pairs in advance, the model was trained to control the steering wheel, accelerate, and brake by imitating experts. The DeepMind team also used behavior cloning to initialize the policy network when training AlphaGo [23]. They encoded each chessboard state as a tensor through an encoder, and trained the policy network through supervised learning. This method greatly improves the accuracy of the model placing chess pieces and speeds up the training speed. Some other works related to behavior cloning can be found in [24, 25].

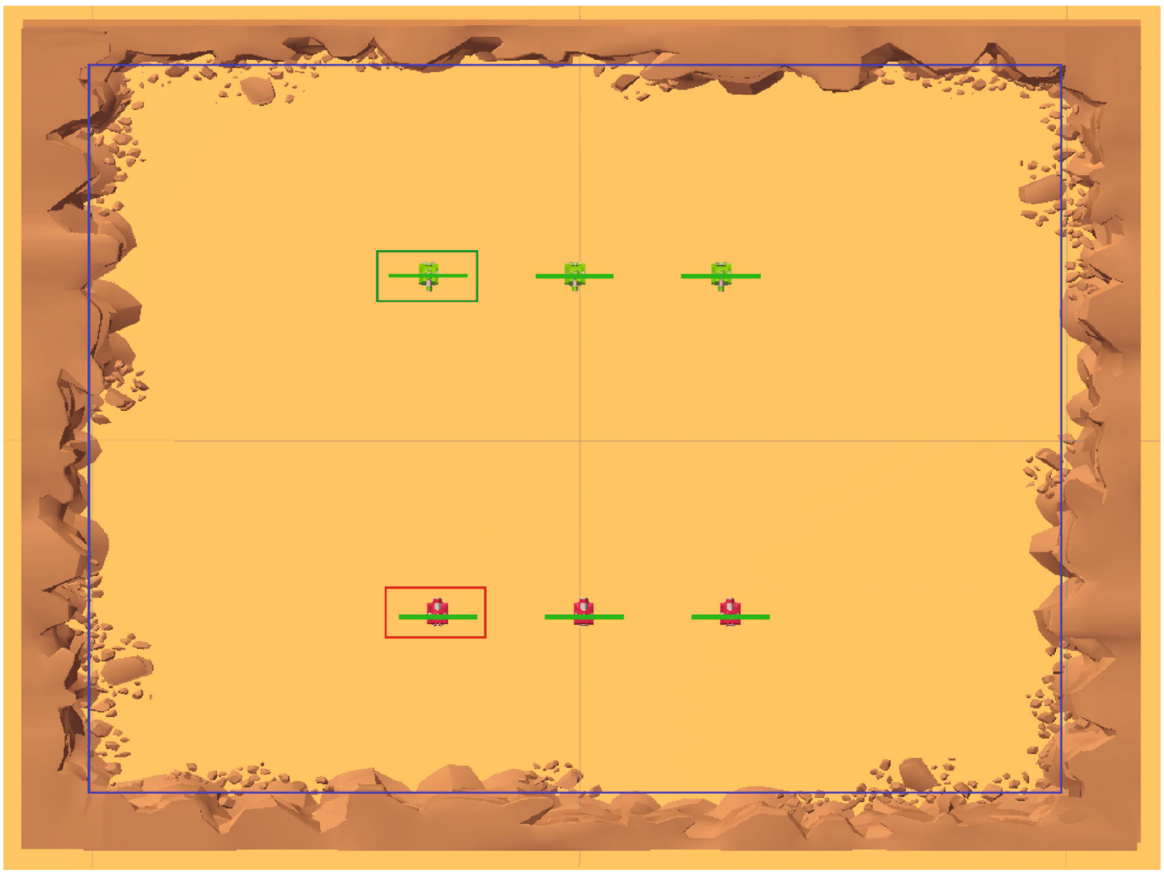
In this paper, we build a 3 V3 swarm fighting simulation environment based on the Unity platform, and conducted performance evaluation of two MARL based training methods for swarm fighting. In particular, for the first method, we directly employ the MARL training method, and for the second method, we combine the MARL training method and the behavior cloning training method by appropriately weighting between these two methods, leading to the combined MARL and behavior cloning (MARL-BC) training method. The behavior cloning expert adopted in this paper is taken from some well-trained model in the final steady phase by the MARL training method. From the perspective of winning rate, the performances of these two different training methods can be divided into three phases. In the first phase, learning progresses slowly for both these two training methods. The reason is obvious for MARL training method. While, for the MARL-BC training method, since the data available for learning is mainly for opponents of comparable ability, for the scenarios where the opponent is weak, these data could not bring any extra benefits. As the model by the MARL training method grows stronger, the experience of the behavior cloning expert gradually becomes useful, and the second phase kicks off where the MARL-BC training method takes obvious advantage. Surprisingly, the advantage of the MARL-BC training method will disappear as the learning progress goes on. The reason behind this is that in the final phase, the model trained by the MARL training method still keeps learning from the environment and its opponent, while the model trained by the MARL-BC training method has to follow the rules of the expert to some extent. Therefore, the resulted strategy cannot effectively adapt to the ever changing environment and opponent.

2. Environment and Agent Setting

In this paper, we build the 3 V3 swarm fighting simulation environment based on the Unity platform. Unity is a platform for real-time 3D interactive content creation and operation, which has been widely used in game development, art, architecture, automobile design, and other fields. Also, Unity has a powerful physics engine based on NVIDIAphysX or Havok and an AAA-grade image rendering engine. Based on the above engines, Unity can simulate rigid bodies, particles, and other realistic physical environments with high accuracy. Most importantly, Unity enables fast distributed simulation. When tasks require fast simulation speed other than the frame rate of the rendering process, Unity can simulate from the code level and significantly improve the learning efficiency of RL.

2.1. Environment Setting. Based on the Unity platform, the simulation environment can support a large number of agents to train in a single environment. Figure 1 shows the top and side views of the specific environment. The environment is bounded by boundaries 200 meters long and 160 meters wide, including six intelligent tanks, three in the red teams and three in the green teams. These intelligent tanks have the same attributes and will be referred to as agents later. The agents in both teams have certain endurance and fire limits. The goal of each team is to annihilate the other with maximal efficiency and minimal loss.

The winning condition for each team is to annihilate all agents of the other team. A time limit is added to avoid endless battles. If the time limit is exceeded, the game will be tied. The specific rules are given by Algorithm 1, where Red.num and Green.num represent the number of surviving agents in red team and green team, respectively, and

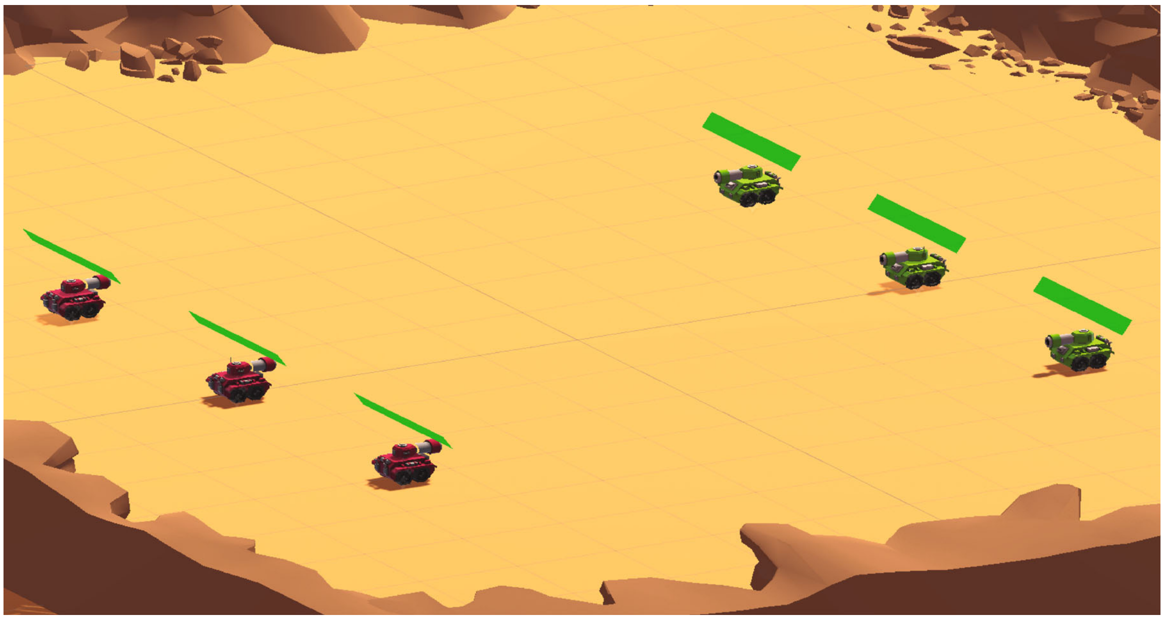


🟩 Agent of green team

🟥 Agent of red team

🟦 Border

(a) Top view of the simulation environment



(b) Side view of the simulation environment

Figure 1: Top and side views of the swarm fighting simulation environment.

```
Input: Begin//bool value, initialize the environment, whether to start round.
Output: Winner of this round.
  Execute this program each frame.
  MaxEnviormentStep =8000//Defines the maximum.
  number of steps in the environment.
  for i≤ MaxEnviormentStep && Begin do.
      ++i.
      if Red.num ==0 && Green.num ==0 then
          return tie
      else if Red.num ==0 then
          return Green.win
      else
          return Red.win
return null
```

Algorithm 1: Global Rules

Green.win (Red.win) means green team (red team) wins the fighting. An independent script acts as a judge throughout the environment, which analyzes each frame in the environment to ensure that the execution of global rules is correct.

2.2. Agent Setting. Each agent has two continuous actions: forward, turn, and a discrete action: fire or not. The agent obtains environmental information by communicating with the surrounding agents through sensors. Now we introduce the basic configuration information of the agent.

2.2.1. Motion. According to the task requirements, the agent can only move in the XOZ plane in the global coordinate system, and the motion control of the agent follows the second-order unicycle model, which is given by

$$\begin{bmatrix} \phi(k+1) \\ x(k+1) \\ y(k+1) \end{bmatrix} = \begin{bmatrix} x(k) + v(k) \cdot \cos(\phi(k)) \cdot T_s \\ y(k) + v(k) \cdot \sin(\phi(k)) \cdot T_s \\ \phi(k) + \omega(k) \cdot T_s \end{bmatrix}, \quad (1)$$

where $(x(k), y(k))$ denote the Cartesian coordinate of the agent; $\phi(k)$ denotes the anticlockwise angle of the agent with respect to the x-axis; $v(k)$ and $\omega(k)$ denote the linear and angular velocities of the agent, respectively; T_s denotes the sampling time. Moreover, we impose limits for linear and angular velocities for each agent. In particular, $v(k) \in [-10m/s, (20m/s)]$, and $\omega(k) \in [-10°/s, (10°/s)]$.

2.2.2. Perception. The agent has three ways to perceive the surrounding environment, i.e., by Raycast Observations, by communication with teammates, and by self-observation. As shown in Figure 2(a), an omnidirectional laser detector is employed by each agent to detect the surrounding obstacles, where 14 rays with a length of 35 m are emitted from the agent to the surroundings to form an omnidirectional laser detector. Each ray returns an n-dimensional vector, where n is the number of detected tags. In this paper, we set the tags to be Wall, Enemy, and Teammate. In Figure 2(b), we can see that the 5th ray detects the enemy, the 8th ray detects the teammate, and the 10th-12th rays detect the wall.

In the simulation environment, we suppose there is regular communication between agents of the same team. By communication, the agents can share information with each other, including HP, velocity, and position, as shown in Table 1.

The agent can obtain the information of the opponents within a certain range that are not blocked by obstacles through observation. If the opponent is behind the agent, there is a 20% chance that the opponent's information can be obtained. The opponent's speed and HP information are summarized in Table 2.

2.2.3. Attack Capability. The agent can attack the opponent by firing projectiles. When the agent chooses to shoot, the projectile acquires an initial angle and an initial velocity relative to the agent's coordinate system. In the simulation environment, the air resistance is ignored, and thus the projectile's trajectory in the air is parabolic subject to gravity. When the projectile hits any object, it will explode, causing damage to all the agents within the blast radius. Let D_i represent the distance between agent i and the projectile, ζ represent the maximum damage of the projectile, η represents the blast radius. Then the damage H_i caused by the projectile to agent i is described as follows:

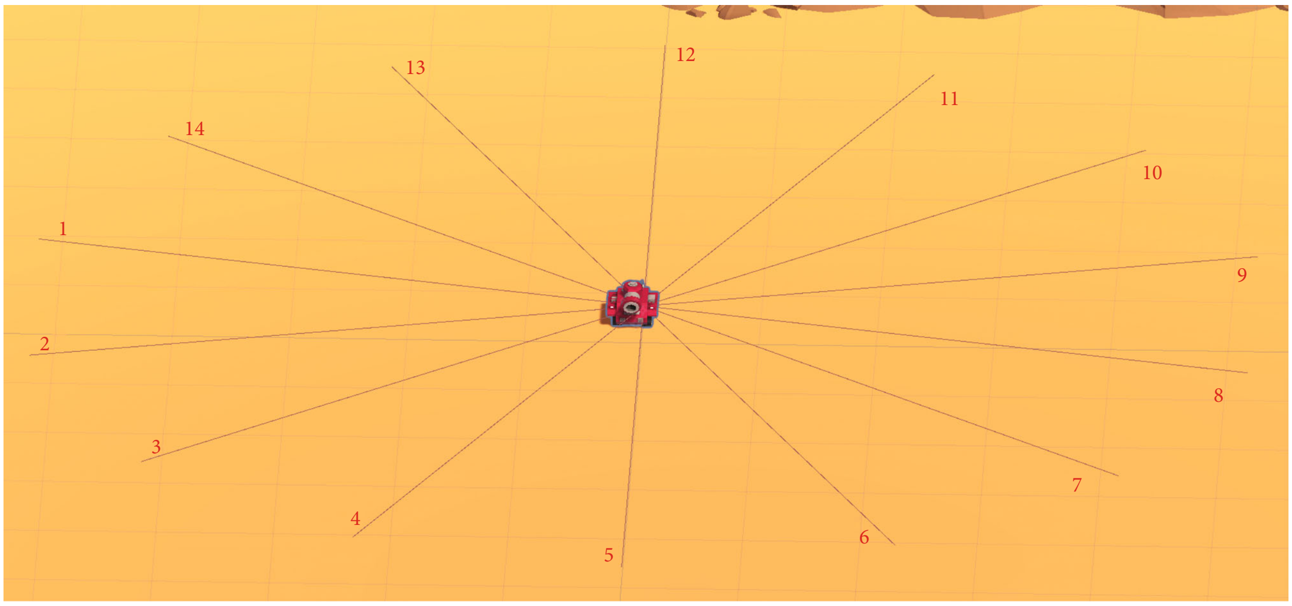
$$H_i = \begin{cases} \zeta^* \left(\dfrac{\eta - D_i}{\eta} \right) & D_i \le \eta \\ 0 & D_i > \eta \end{cases}. \quad (2)$$

The initial velocity at which an agent fires a projectile is controllable. If an agent fires a shell when the opponent is close to it, the shell may injure itself.

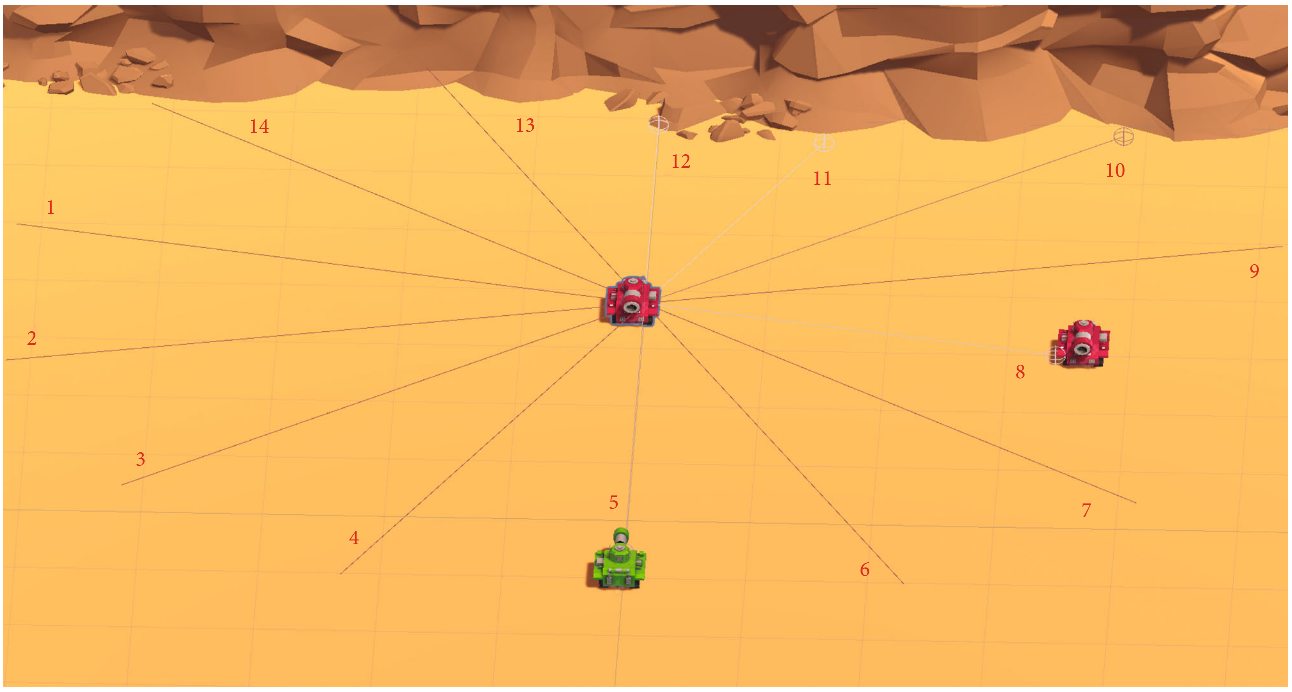
2.3. Reward Setting. In the process of RL training, by setting appropriate positive and negative rewards, the agent can be guided to learn the correct behavior. In the simulation environment, there are two kinds of rewards, agent-wise and group-wise. Agent rewards are used to optimize individual behaviors, so that agents can learn basic behaviors such as attack or retreat. Group rewards are used to encourage cooperative behaviors of group members. The specific positive and negative rewards are given in Tables 3 and 4. The terminologies in the tables are explained as follows:

(i) *HP* represents the remaining blood volume of the agent, and *FullHP* represents the full blood volume of the agent. *HP_Sum* represents the total *HP* of a team

(ii) *ResetTimer* denotes the current number of steps, and *MaxEnvironmentSteps* denotes the longest number of steps in a round. If the number of steps is exceeded, the fight will be forced to end



(a) The sensor of the agent



(b) Detection by the sensor

FIGURE 2: The working principle of the sensor.

TABLE 1: Information obtained by communication.

Number of feature	Meaning of feature
1	The remaining HP of the agent
2	The X-coordinate of the agent
3	The Z-coordinate of the agent
4	The X-direction velocity of the agent
5	The Z-direction velocity of the agent

TABLE 2: Information obtained by self-observation.

Number of feature	Meaning of features
1	The remaining HP of the opponent
2	The X-direction velocity of the agent
3	The Z-direction velocity of the agent

Table 3: Positive reward setting.

Type	Situation	Reward
Agent	Projectile hits opponent	$0.1 * \left(\frac{H_i}{\zeta}\right) + 0.15 * \left(\frac{\mathrm{HP}}{\mathrm{FullHP}}\right)$
	Group wins	$0.6 + \left(\frac{\mathrm{HP}}{\mathrm{FullHP}}\right) * \left(1 - \left(\frac{\mathrm{ResetTimer}}{\mathrm{MaxEnvironmentSteps}}\right)\right) * 0.6$
	Opponent is killed	$0.15 * \left(\frac{H_i}{\zeta}\right) + 0.2 * \left(\frac{\mathrm{HP}}{\mathrm{FullHP}}\right)$
Group	Opponent is killed	0.1
	Group wins	$0.9 - 0.5 * \left(\left(\frac{\mathrm{ResetTimer}}{\mathrm{MaxEnvironmentSteps}}\right) + 0.2 * (HP_{\mathrm{Sum}})\right)$

Table 4: Negative reward setting.

Type	Situation	Reward
Agent	Agent is killed	$-0.2 + \left(\frac{\mathrm{ResetTimer}}{\mathrm{MaxEnvironmentSteps}}\right) * 0.1$
	Group failed	-0.1
	Injure teammates	$\left(-0.2 * \left(\frac{H_i}{\zeta}\right)\right)$
Group	Group ties	$-0.2 + \left(\frac{\mathrm{ResetTimer}}{\mathrm{MaxEnvironmentSteps}}\right) * 0.12$
	Group fails	$-0.8 + \left(\frac{\mathrm{ResetTimer}}{\mathrm{MaxEnvironmentSteps}}\right) * 0.5$

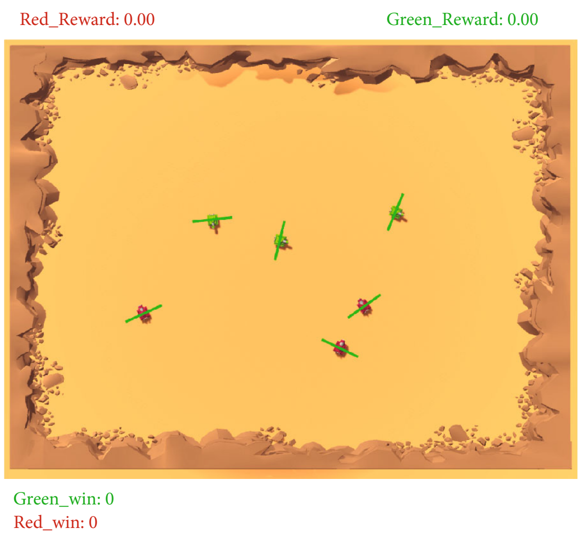
3. Algorithm

RL is a trial-and-error-based machine learning method. During the learning process, the agent is not told what actions to take, but must try to discover which action may produce more benefits. Let s_t represent the state of the agent at step t, a_t represent the action taken by the agent at step t, and r_{t+1} represent the reward obtained by the agent performing action a_t in state s_t. The agent uses the reward of each step as feedback to adjust the weighted parameters of the reinforcement learning network to maximize the reward, and the agent also learns the optimal strategy at each step. In the process of MARL, in order to strengthen the cooperation between agents, the rewards of the environment are mostly shared rewards. We want the agent's behavior to maximize the group's future reward. However, during the training process, the agent may terminate early and be removed from the environment. The removed agent cannot learn the success or failure of the group from subsequent phase. This problem is known as the postmortem credit allocation problem. The MA-POCA algorithm proposed by the Unity team solves this problem very well [26]. The algorithm introduces an attention mechanism at the input of RL, so that the network has the ability to deal with a variable number of agents, which is convenient for us to add or delete agents during the training process. In this experiment, we choose this algorithm to train the agent.

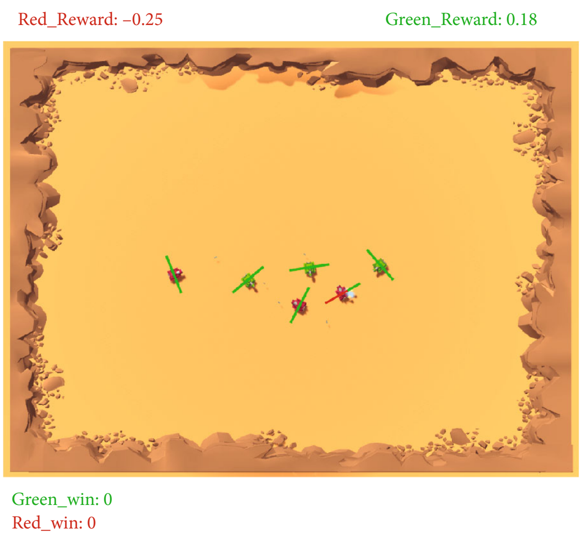
Table 5: Hyperparameters used for this experiment.

Type		MARL	MARL-BC
Hyperparameters	Batch size	1024	1024
	Buffer size	20480	20480
	Learning rate	0.0001	0.0001
	Entropy bonus	0.005	0.005
	Num epoch	3	3
Network settings	Hidden units	512	512
	Num layers	3	3
Reward signals	Discount factor	0.99	0.99
	Strength	1.0	1.0
Behavior cloning	Steps	/	100 M
	Strength	0.5	0.5

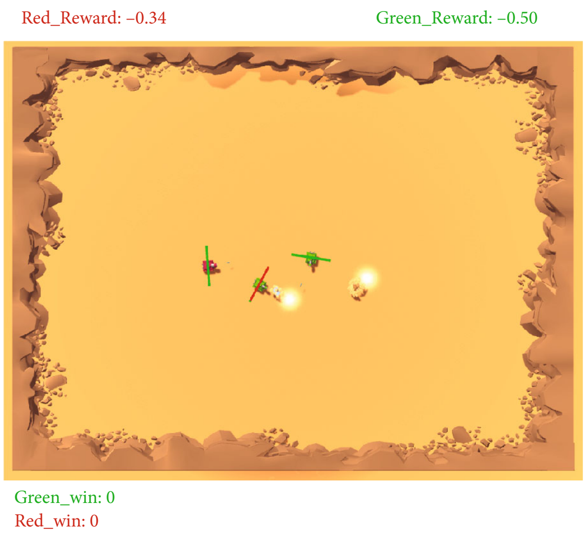
In the experiment, we set up two groups of experiments. One group of experiments only use MA-POCA algorithm, and the other group of experiments use MA-POCA and behavioral cloning. In behavioral cloning, the most important factor is the demo model. The demo of this experiment comes from the agent with the highest winning rate obtained from many experiments. We train 100 M times to get the agent model through MARL by self-play. At this far, the capability of the agent reaches a bottleneck. We set the



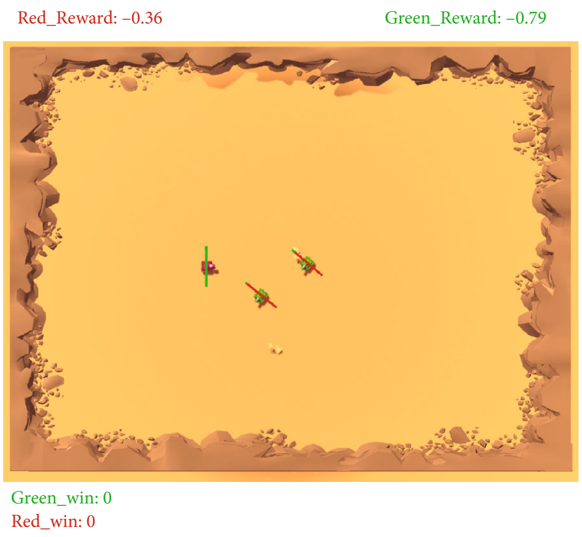
(a) Swarm fighting starts



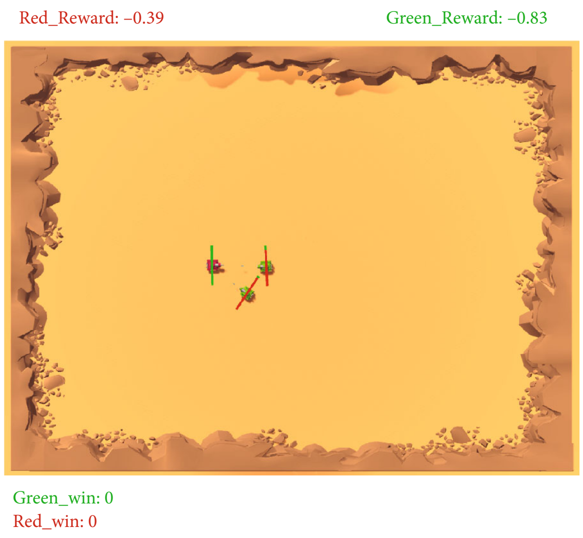
(b) Green team is attacking



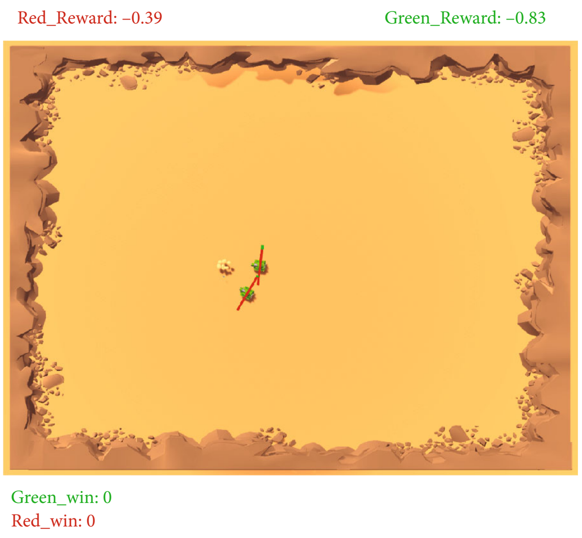
(c) Three agents are killed



(d) Three agents remain



(e) The green team surrounds the red team



(f) The green team wins

Figure 3: The whole process of a single round of swarm fighting.

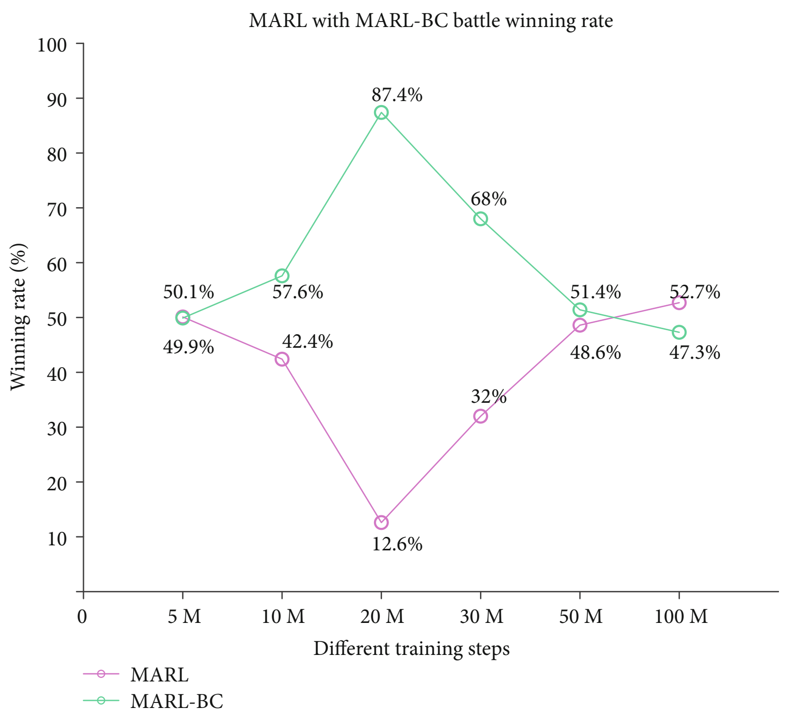


Figure 4: Winning rate of different training methods.

adversary with stronger attributes in terms of speed, HP, angular velocity, and field of view. By gradually strengthening the properties of the adversary, the ability of the agent is gradually improved. Finally, we obtain an agent model with a higher winning rate. By recording the behavior of the agent during testing, we obtain the demo files used for behavioral cloning training. In the initial stage for the MARL-BC method, the agent's decision is made equally based on the knowledge from RL and from the demo files of the expert by the behavioral cloning training method. This training process will continue for 100 M steps.

The experiment is conducted in the ML-Agent development environment of Unity, where the version number is 0.28.0. The hyperparameters used in this experiment are listed in Table 5, and those parameters not listed in the table just take default values. The terminologies of Table 5 are explained as follows:

(i) Hyperparameters

Batch Size Setting Batch Size to 1024 means collecting 1024 data samples at one time for training. Generally speaking, small batch size will increase the randomness of the gradient descent, thus making the algorithm difficult to converge. On the other hand, increasing batch size can reduce sample randomness, improve learning efficiency, and make the direction of the gradient descent more stable.

Buffer Size refers to the amount of experience to collect before updating the policy model. For the swarm fighting of three agents, setting Buffer Size to be 20480 can help the agents learn quickly.

Learning rate corresponds to the magnitude of each gradient descent update step. Excessive learning rate will violate the stability of the training process. Extensive experiments show that the model considered in this paper has better performance when the learning rate is set to be 0.0001.

Entropy Bonus encourages agent to take unpredictable actions. Setting Entropy Bonus to be 0.005 can help the agent improve the generalization ability in the training process.

Num Epoch denotes the number of times to pass through the experience buffer when performing gradient descent optimization. Decreasing Num epoch will make the updates more stable, but in the meantime slows down the training speed. To balance stability and training efficiency, we set Num epoch to be 3.

(ii) Network Settings

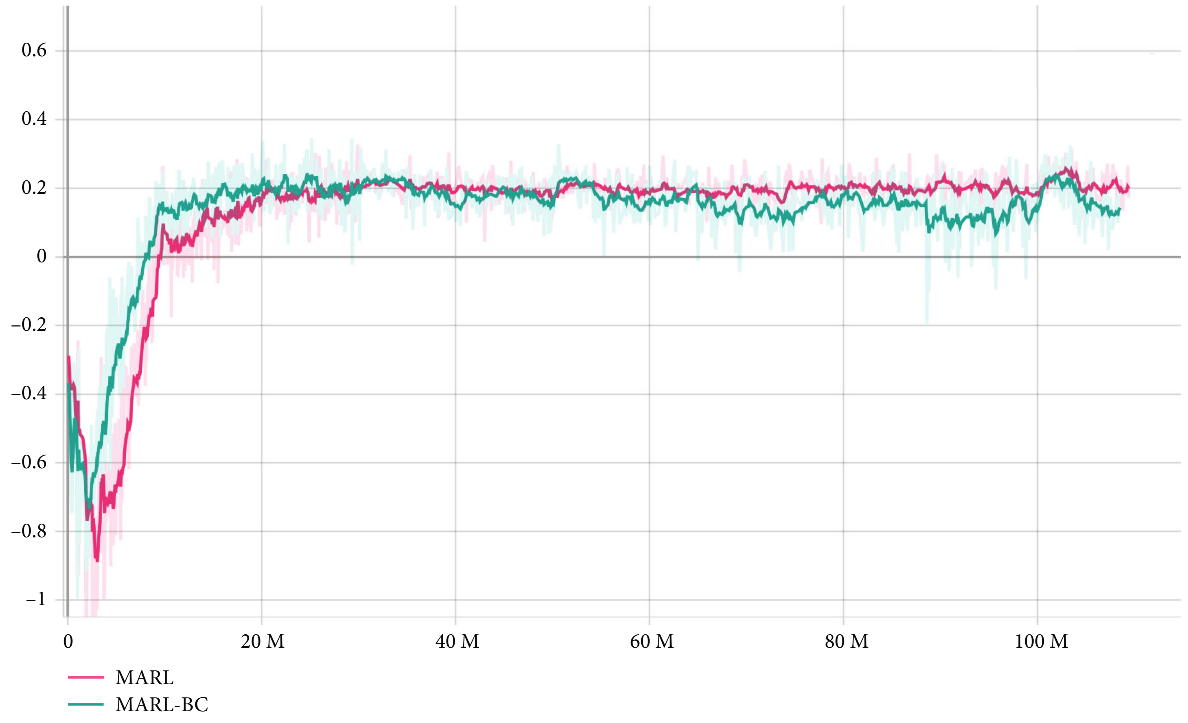
Hidden Units represent the neural network units of the hidden layer, ranging from 32 to 512 as recommended by the official document. In this paper, the number of hidden units is set to be 512 to maximize the ability of the agents.

Num layers refers to the number of hidden layers of the neural network. Taken into account the complex situation of swarm fighting, we set Num layers to be 3.

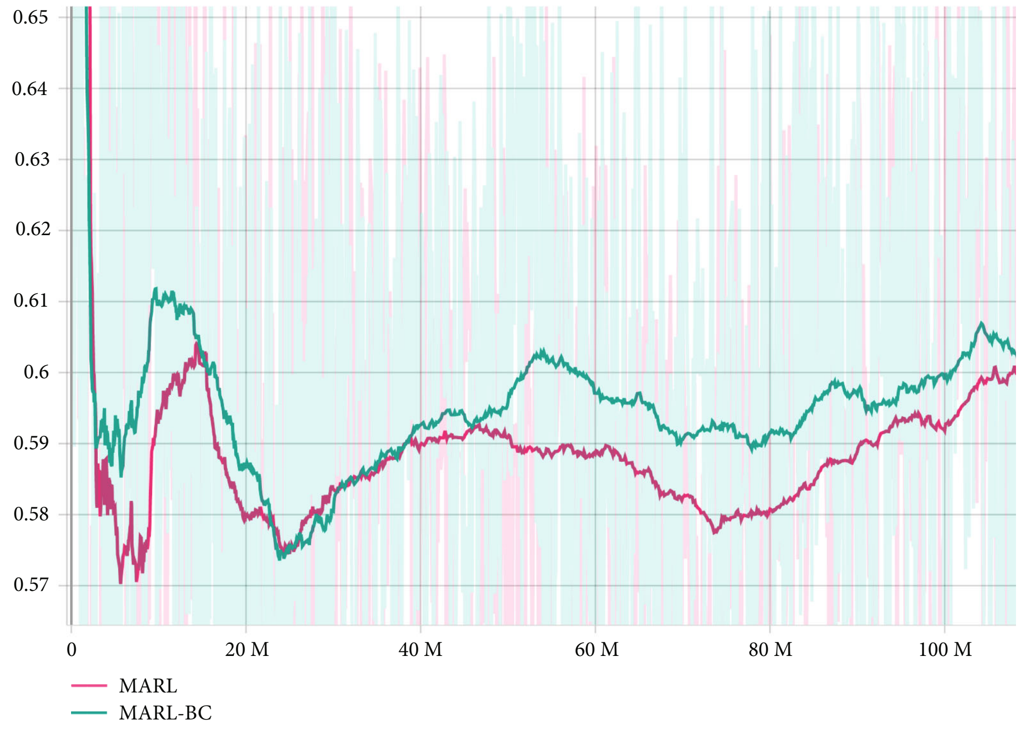
(iii) Reward Signals

Discount Factor denotes the discount factor for future rewards coming from the environment. It reflects how far into the future the agent should care about the possible rewards. Since the swarm fighting problem is a long sequence process, we set Discount Factor to be 0.99.

Strength represents the multiplication coefficient of the original reward. It is set to be 1 by the official recommendation.



(a) Cumulative agent reward throughout the training process



(b) Cumulative group reward throughout the training process

Figure 5: Cumulative reward of agent and group.

(iv) Behavior Cloning

Steps represent the training steps of behavior cloning, which are set to be 100 M. After 100 M steps, the model will no longer be trained by the behavior cloning method.

Strength represents the proportion of behavior cloning training in the training process. In this paper, it is set to be 0.5, i.e., the agents will be trained equally by the MARL and the behavior cloning training method.

Two indicators are adopted to evaluate the performance of the algorithm, namely, the winning rate and the cumulative reward. We conduct ten sets of parallel tests of the same environment to speed up the test efficiency. To make the experimental results fair, we set up two identical competitions in each set of test where the two sides of competitors shall switch their initial status. In order to make the experimental results more convincible, we measure the winning rate of the model through at least 2000 rounds of tests. As mentioned above, the agent takes different actions in different states. The system gives different rewards according to the quality of the action. If the average reward value of the agent is large, it means that the agent makes more correct decisions, and the training effect of the agent is better. Figure 3 shows the whole process of a single round of swarm fighting.

4. Results

The results are shown by Figures 4 and 5. Figure 4 shows the winning rate of MARL and MARL-BC based training method at different training steps. MARL refers to the training method by MA-POCA, which is represented by the pink line. MARL-BC refers to the training method by MA-POCA plus behavior cloning, which is represented by the green line.

At 5 M steps, the winning rates of both sides are similar. Starting from 5 M steps, the winning rate of MARL-BC gradually increases until it reaches 87.4% at 20 M steps. After that, the winning rate of MARL-BC starts to decline. At 50 M steps, the winning rates of the two training methods are close to each other again, where the winning rate of MARL-BC is only 2.8% better than that of MARL. When the training continues up to 100 M steps, the winning rate of MARL is 52.7%, which has exceeded the winning rate of MARL-BC.

Figure 5(a) shows the profile of the cumulative reward throughout the whole training process. In the early training phase, the cumulative reward of MARL-BC is almost the same as that of MARL, and both of which show a downward trend. As the experiment continues, the cumulative reward of MARL-BC first reaches the inflection point and starts to increase, ahead of the cumulative reward of MARL. The cumulative reward of MARL reaches its inflection point at 3 M and also starts to increase, but it is slower than MARL-BC. The cumulative reward of MARL-BC keeps being higher than that of MARL until the experiment reaches 22 M steps, at which the two cumulative rewards of the two training method are equal. In the next period, the cumulative rewards of MARL-BC and MARL are comparable. When the experiment reaches 54 M steps, the cumulative reward of MARL begins to lead MARL-BC, which continues until the end of the experiment. Figure 5(b) shows the profile of the group reward throughout the whole training process. Similar to Figure 5(a), MARL-BC reaches the inflection point first, but the difference is that as the training progresses, the group reward of the MARL-BC method is larger than that of MARL.

As described above, behavior cloning helps agents learn quickly to fight against opponents. While, behavior cloning helps the agent beat the agent trained with a certain number of steps. When faced with an agent with low intelligence which is insufficiently trained, behavior cloning cannot help much. Therefore, at 50 M steps, the number of training steps of the agent of MARL is not enough, and MARL-BC cannot provide useful experience. At this phase, MARL-BC and MARL have almost the same winning rate. As the experiments proceeded, the model using the MARL method became more sophisticated. Prior knowledge of behavioral cloning comes into play, helping agents using MARL-BC progress rapidly and maintain a high winning rate. In the final phase of the experiment, the agent using the MARL method learns from wrong decisions and keeps updating the model, and thus the ability of the agent is still growing. However, for the behavior cloning method, due to the limited scenarios covered by prior knowledge, the strategies provided by prior knowledge are not necessarily optimal, hindering the model's progress to some certain extent. The winning rate gap between the two method starts to narrow down, and finally the winning rate of the agent using the MARL method surpassed that of the agent using MARL-BC. If the training continues, the results are not difficult to predict that the advantage of the agent trained using the MARL method will become larger, and the winning rates of the two will converge with a steady difference.

5. Conclusion

In this paper, we evaluate the performance of two multiagent reinforcement learning based training methods, namely, MARL and MARL-BC. Two performance evaluation indicators are adopted, i.e., the winning rate and the cumulative reward. By comprehensive experiments, the following results are revealed.

(i) To some extent, behavior cloning can help agents learn strategies quickly to deal with specific opponents, which enables agents to defeat their opponents before and during training with extra knowledge and experience. However, as the number of the training steps keeps growing, the knowledge and experience from the behavior cloning may prevent the modeling from improving compared with the MARL training method which, for all the training process, keeps learning from its opponent and the environment

(ii) Because of the limitations of behavioral cloning training method in the later phase of training, it should be considered as an auxiliary training method in the prephase of the training process, rather than a training method for the whole training process

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This research was funded in part by the National Natural Science Foundation of China under grant number 62173149, 62276104, and in part by the Guangdong Natural Science Foundation under grant number 2021A1515012584, 2022A1515011262.

References

[1] C. J. C. H. Watkins, *Learning from delayed rewards*, King's College, Cambridge United Kingdom, 1989.

[2] D. P. Bertsekas and J. N. Tsitsiklis, "Neurodynamic programming: an overview," in *Proceedings of 1995 34th IEEE conference on decision and control*, vol. 1, pp. 560–564, New Orleans, LA, USA, 1995.

[3] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *International conference on machine learning. PMLR*, vol. 32, no. 1, pp. 387–395, Beijing, China, 2014.

[4] C. Huang, S. Lucey, and D. Ramanan, "Learning policies for adaptive tracking with deep feature cascades," in *Proceedings of the IEEE international conference on computer vision*, pp. 105–114, Venice, Italy, 2017.

[5] K. Yu, C. Dong, L. Lin, and C. C. Loy, "Crafting a toolchain for image restoration by deep reinforcement learning," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2443–2452, Salt Lake City, UT, USA, 2018.

[6] H. Xu, S. Feng, Y. Zhang, and L. Li, "A grouping-based cooperative driving strategy for cavs merging problems," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 6, pp. 6125–6136, 2019.

[7] M. Tan, "Multi-agent reinforcement learning: independent vs. cooperative agents," in *Proceedings of the tenth international conference on machine learning*, pp. 330–337, Amherst, MA, USA, 1993.

[8] S. Bajpai, "Application of deep reinforcement learning for Indian stock trading automation," 2021, https://arxiv.org/abs/2106.16088.

[9] K.-F. Tang, H.-C. Kao, C.-N. Chou, and E. Y. Chang, "Inquire and diagnose: neural symptom checking ensemble using deep reinforcement learning," *NIPS Workshop on Deep Reinforcement Learning*, 2016.

[10] J. Wang, T. Shi, Y. Wu, L. Miranda-Moreno, and L. Sun, "Multi-agent graph reinforcement learning for connected automated driving," in *Proceedings of the 37th International Conference on Machine Learning (ICML)*, pp. 1–6, Vienna, Austria, 2020.

[11] J. Wu, Z. Huang, W. Huang, and C. Lv, "Prioritized experience-based reinforcement learning with human guidance: methdology and application to autonomous driving," 2021, https://arxiv.org/abs/2109.12516.

[12] J. Luketina, N. Nardelli, G. Farquhar et al., "A survey of reinforcement learning informed by natural language," 2019, https://arxiv.org/abs/1906.03926.

[13] O. Vinyals, I. Babuschkin, W. M. Czarnecki et al., "Grandmaster level in StarCraft II using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[14] M. L. Littman, "Markov games as a framework for multi-agent reinforcement learning," in *Machine learning proceedings 1994*, pp. 157–163, New Brunswick, New Jersey, USA, 1994.

[15] R. Lowe, Y. I. Wu, A. Tamar, J. Harb, O. Pieter Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," *Advances in neural information processing systems*, vol. 30, 2017.

[16] H. He, J. Boyd-Graber, K. Kwok, and H. Daume, "Opponent modeling in deep reinforcement learning," in *International conference on machine learning PMLR*, vol. 48, pp. 1804–1813, New York City, NY, USA, 2016.

[17] B. Peng, T. Rashid, C. Schroeder de Witt et al., "Facmac: factored multi-agent centralised policy gradients," *Advances in Neural Information Processing Systems*, vol. 34, pp. 12 208–12 221, 2021.

[18] J. Su, S. Adams, and P. Beling, "Valuedecomposition multi-agent actor-critics," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35no. 13, pp. 11 352–11 360, 2021.

[19] S. Iqbal and F. Sha, "Actor-attention-critic for multi-agent reinforcement learning," in *International conference on machine learning. PMLR*, pp. 2961–2970, Los Angeles, USA, 2019.

[20] P. Peng, Y. Wen, Y. Yang et al., "Multiagent bidirectionally-coordinated nets: emergence of human-level coordination in learning to play starcraft combat games," 2017, https://arxiv.org/abs/1703.10069.

[21] D. Rengarajan, G. Vaidya, A. Sarvesh, D. Kalathil, and S. Shakkottai, "Reinforcement learning with sparse rewards using guidance from offline demonstration," 2022, https://arxiv.org/abs/2202.04628.

[22] F. Codevilla, M. Muller, A. Lopez, V. Koltun, and A. Dosovitskiy, "End-to-end driving via conditional imitation learning," in *2018 IEEE international conference on robotics and automation (ICRA)*, pp. 4693–4700, Brisbane, QLD, Australia, 2018.

[23] D. Silver, A. Huang, C. J. Maddison et al., "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[24] F. Codevilla, E. Santana, A. M. Lopez, and A. Gaidon, "Exploring the limitations of behavior cloning for autonomous driving," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 9329–9338, Seoul, Korea, 2019.

[25] Y. Chen, J. Su, and W. Wei, "Multi-granularity textual adversarial attack with behavior cloning," 2021, https://arxiv.org/abs/2109.04367.

[26] A. Cohen, E. Teng, V.-P. Berges et al., "On the use and misuse of absorbing states in multi-agent reinforcement learning," 2021, https://arxiv.org/abs/2111.05992.