electronics

MDPI

Article

The Study of Crash-Tolerant, Multi-Agent Offensive and Defensive Games Using Deep Reinforcement Learning

Xilun Li [1], Zhan Li [2,3]*, Xiaolong Zheng [3], Xuebo Yang [3] and Xinghu Yu [4]

[1] School of Astronautics, Harbin Institute of Technology, Harbin 150001, China
[2] Department of Mathematics and Theories, Peng Cheng Laboratory, Shenzhen 518000, China
[3] Research Institute of Intelligent Control and Systems, Harbin Institute of Technology, Harbin 150001, China
[4] Ningbo Institute of Intelligent Equipment Technology Co., Ltd., Ningbo 315201, China
* Correspondence: zhanli@hit.edu.cn

Abstract: In the multi-agent offensive and defensive game (ODG), each agent achieves its goal by cooperating or competing with other agents. The multi-agent deep reinforcement learning (MADRL) method is applied in similar scenarios to help agents make decisions. In various situations, the agents of both sides may crash due to collisions. However, the existing algorithms cannot deal with the situation where the number of agents reduces. Based on the multi-agent deep deterministic policy gradient (MADDPG) algorithm, we study a method to deal with a reduction in the number of agents in the training process without changing the structure of the neural network (NN), which is called the frozen agent method for the MADDPG (FA-MADDPG) algorithm. In addition, we design a distance–collision reward function to help agents learn strategies better. Through the experiments in four scenarios with different numbers of agents, it is verified that the algorithm we proposed can not only successfully deal with the problem of agent number reduction in the training stage but also show better performance and higher efficiency than the MADDPG algorithm in simulation.

Keywords: multi-agent deep reinforcement learning; offensive and defensive game; frozen agent method

Citation: Li, X.; Li, Z.; Zheng, X.; Yang, X.; Yu, X. The Study of Crash-Tolerant, Multi-Agent Offensive and Defensive Games Using Deep Reinforcement Learning. *Electronics* **2023**, *12*, 327. https://doi.org/10.3390/electronics12020327

Academic Editor: Dah-Jye Lee

Received: 28 November 2022
Revised: 23 December 2022
Accepted: 6 January 2023
Published: 8 January 2023

1. Introduction

Cooperation and competition as well as pursuit and evasion are common elements and behaviors in nature. The competitive or cooperative game evolved from them and has been a hot topic for many scholars to study for a while [1–4]. Some scholars have studied the ODG strategies in different confrontation scenarios. In the attack and escort scenarios of UAV swarms, Zou et al. [5] designed the application of bilateral cooperation strategies between the escort swarms and the high-value target to form different confrontation and avoidance strategies. In the coastal defense system, Zhang et al. [6] proposed a system design scheme based on layered distributed multi-agents, which effectively improved the decision-making efficiency and interception success rate of the system. There are also some scholars who applied deep reinforcement learning (DRL) algorithms to battlefield decision making. Yang et al. [7] proposed a PPO algorithm-based UAV air combat maneuver decision method and conducted close-range air combat simulation to verify the effectiveness of the algorithm. Zhao et al. [8] applied the deep reinforcement learning algorithm to the military field and proposed an end-to-end collaborative planning method for intelligent reconnaissance missions of dual UAVs.

As the multi-agent scenarios become more and more complex, some advanced machine learning algorithms are applied to solve such problems. However, the traditional DRL algorithms have many disadvantages, such as instability in the multi-agent environment [9]. Therefore, some MADRL algorithms came into being, and they have been widely used in various fields [10]. Qi et al. [11] applied MADRL algorithms to Wireless Local Area Networks (WLANs) and proposed an On-Demand Channel Bonding (O-DCB) algorithm for heterogeneous WLANs to reduce transmission delay. Jung et al. [12] proposed a novel

coordinated MADRL algorithm for energy sharing among multiple UAVs in order to conduct big data processing in a distributed manner. In the field of smart ships, Chen et al. [13] proposed a cooperative collision avoidance approach for multiple ships using an MADRL algorithm based on the DQN method.

For complex competitive or cooperative games, the analytical Nash equilibrium solution cannot be solved mathematically [14]. As one of the representatives of the MADRL algorithms [15], the MADDPG algorithm is widely used to solve similar problems. The MADDPG algorithm extends the deep deterministic policy gradient (DDPG) algorithm to multi-agent environments. The MADDPG algorithm assumes that each agent has its independent critic network and actor network and that each agent has its independent return function. In this way, the MADDPG algorithm can simultaneously solve the multi-agent problem in a collaborative and competitive environment.

In the pursuit–evasion game, Wan et al. [16] proposed a novel adversarial attack trick and adversarial learning MADDPG algorithm to help agents learn strategies. Experimental results verified that the proposed approach provided superior performance. Lei et al. [17] artificially designed a rule-coupled method based on the MADDPG algorithm to improve the confrontation ability and combat efficiency in a drone swarm attack and defense confrontation. In [18], based on the MADDPG algorithm, preferential experience replay was used to achieve good results in multi-robot path planning. In [19], a multi-agent coronal bidirectionally coordinated with target prediction network (CBC-TP Net) was constructed in the pursuit game scenario of a UAV swarm. Its performance was better than that of the MADDPG algorithm. In many practical application scenarios, the number of agents changes with the training. However, due to the limitations of the fully connected network in the MADDPG algorithm, the dimensions of the input data cannot be changed. When the number of agents changes, the dimension of the vector input to the Q network will change. To solve this problem, the first layers of the policy and Q network were changed to bidirectional LSTM in [19] to adapt to the change in the number of agents in the training process. However, its disadvantage is that bidirectional LSTM has a complex structure. Specifically, when the input vector dimension is large for multiple agents, the operation is very time-consuming.

An improved MADDPG algorithm called the FA-MADDPG algorithm is proposed in this paper to solve the agent number reduction problem. The method we propose does not need to change the structure of the neural network. It can realize training under the condition that the number of agents changes based on the original fully connected policy network and fully connected Q network. The FA-MADDPG algorithm can not only handle the problem well but also show superior performance and effectiveness for the agents, and all the agents can learn efficient strategies during training.

The outline of this article is as follows. Section 2 introduces the mathematical model of the ODG scenario. Section 3 provides the FA-MADDPG method and the detailed implementation process. Section 4 verifies the effectiveness of the FA-MADDPG method in an agent crash scenario through simulation experiments. Section 5 gives the conclusions and future work.

2. Scenario Description and Modeling

2.1. Scenario Description

This scenario contains defending agents, attacking agents, and one target. As shown in Figure 1, each agent is regarded as a circle with the same radius in a two-dimensional plane. The attacking agents aim to hit the target while evading interception by defending agents. The goal of the defending agents is to cooperate to intercept the attacking agents and protect the target. Each agent does not know the actions or policies of the other agents. The above scenario can be described as an ODG scenario. In order to achieve their respective goals, the whole game involves interception and anti-interception, and some agents may crash due to collisions.

The ODG scenario studied in this paper is different from the pursue-and-escape game, and the number of agents on both sides is different. Therefore, the Hungarian assignment algorithm could not be used to transform the multi-agent ODG scenario into multiple parallel independent chase-and-escape games with one agent on each side. It is a hard problem for agents to make decisions by mathematical derivation. That is why we used the MADRL algorithm to solve this problem.

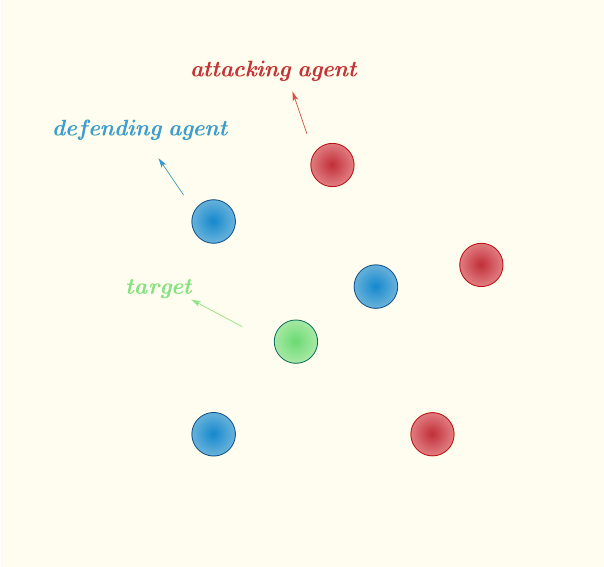

Figure 1. Scenario diagram.

2.2. Agent Modeling

In this article, we only used the circles in the two-dimensional plane for the models in this study. The dynamic and kinematic models of each agent were modeled as follows:

$$\begin{cases} \dot{x} = v_x \\ \dot{y} = v_y \\ \dot{v}_x = \frac{F_x}{m} \\ \dot{v}_y = \frac{F_y}{m} \end{cases} \tag{1}$$

where x and y are the displacements of the agent relative to the target in the x and y directions, respectively, and v_x and v_y are the velocity of the agent relative to the target in the x and y directions, respectively. All the particles have the same mass and are all m. F_x and F_y are the forces of the agent relative to the target in the x and y directions, respectively. In Section 4, we will apply the FA-MADDPG algorithm to train F_x and F_y as the actions of each agent.

3. Frozen Agent Method for MADDPG

3.1. Problem Formulation

In the ODG, we need to control multiple agents on both sides to achieve their goals. This process can be described as the Stochastic Markov Game (SMG) [20], which is a multi-agent extended form of Markov Decision Processes (MDPs) [15,21,22]. An SMG with N agents can be expressed as $\Gamma \triangleq (S, A^1, \ldots, A^N, r^1, \ldots, r^N, p, \gamma)$, where S is the state space, A^j is the action space for agent j, and $r^j : S \times A^i \mapsto \mathbb{R}$ is the reward function for the agent j. The state transition probability p is $S \times A^1 \times \ldots \times A^N \to \Omega(s)$, which describes the random change in state over time . $\Omega(s)$ is the set of probability distribution in the

whole state space. The discount factor is such that $\gamma \in [0,1)$. The goal of each agent is to maximize its total expected reward:

$$R_i = \sum_{t=0}^{T=\infty} \gamma^t r_i^t \tag{2}$$

The goal of the algorithm is to train the optimal strategy $\pi_i(a_i \mid o_i)$,$a_i \in A_i, o_i \in O_i$. a_i and o_i represent the action and observation state of the current time step, respectively.

In order to solve the mixed competitive-cooperative multi-agent ODG problem, the MADRL algorithm is applied in this paper. The main idea of the policy-based DRL approach is to maximize the objective function by directly adjusting the parameter θ in the direction of $\nabla_\theta J(\theta)$. For a game with N agents, its strategy parameter can be expressed as $\theta = \{\theta_1, \ldots, \theta_N\}$. The policy of all agents is represented as $\pi = \{\pi_1, \ldots, \pi_N\}$. If N continuous deterministic strategies μ_{θ_i} (abbreviated as μ_i) are used, then the gradient can be written as

$$\nabla_{\theta_i} J(\mu_i) = \mathbb{E}_{\mathbf{x},a\sim\mathcal{D}}\left[\nabla_{\theta_i}\mu_i(a_i \mid o_i)\nabla_{a_i} Q_i^{\mu}(\mathbf{x}, a_1, \ldots, a_N)\Big|_{a_i=\mu_i(o_i)}\right] \tag{3}$$

where $Q_i^{\mu}(\mathbf{x}, a_1, \ldots, a_N)$ is a centralized action value function, which takes the action $a_1, \ldots, a_N$ of all agents plus the state information $\mathbf{x} = (o_1, \ldots, o_N)$ as input and outputs a value Q for agent i under strategy μ . The experience playback buffer $\mathcal{D}$ contains a tuple $(\mathbf{x}, \mathbf{x}', a_1, \ldots, a_N, r_1, \ldots, r_N)$. The centralized action value function Q_i^{μ} is updated as follows:

$$\begin{aligned} L(\theta_i) &= E_{\mathbf{x},a,r,\mathbf{x}'}\left[\left(Q_i^{\mu}(x, a_1, \ldots, a_N) - y\right)^2\right] \\ y &= r_i + \gamma Q_i^{\mu'}(x', a_1', \ldots, a_N')\Big|_{a_j'=\mu_j'(o_j)} \end{aligned} \tag{4}$$

where $\mu = \left\{\mu_{\theta_1'}, \ldots, \mu_{\theta_N'}\right\}$ is the target policy set with the delay parameter θ_i'. This is the MADDPG method. Its basic idea is centralized training and decentralized execution, as shown in Figure 2.

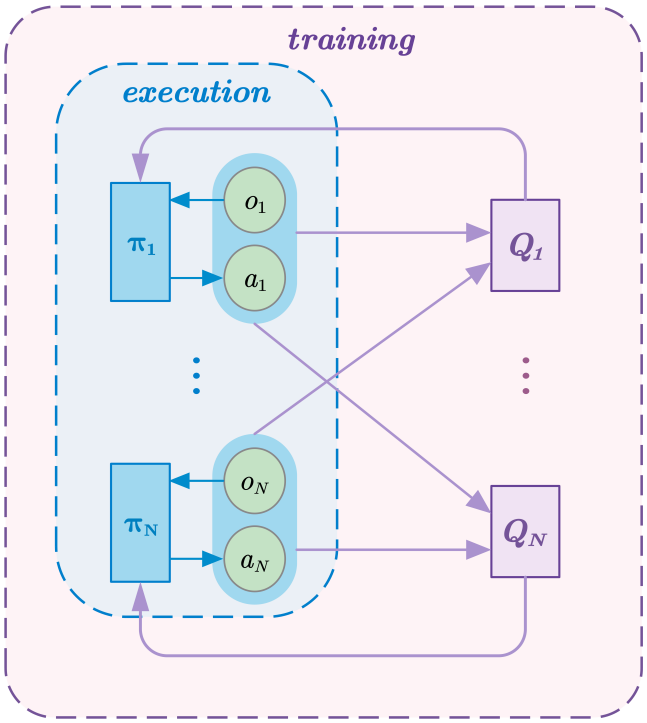

Figure 2. Overview of MADDPG approach.

3.2. Frozen Agent Method

In some practical scenarios, such as the ODG of UAVs, it is assumed that the UAVs on both sides will crash due to collisions during training. Under the centralized training framework of the MADDPG algorithm, the Q network requires the input of all agents' information. If we let the crashed agents continue to interact in the environment during the training process, then the decisions of other agents will inevitably be affected. The FA-MADDPG method we propose aims to solve the above problems. In our proposed approach, after the agent crashes, the agent is frozen and no longer interacts with the environment. The algorithm's structure diagram is shown in Figure 3.

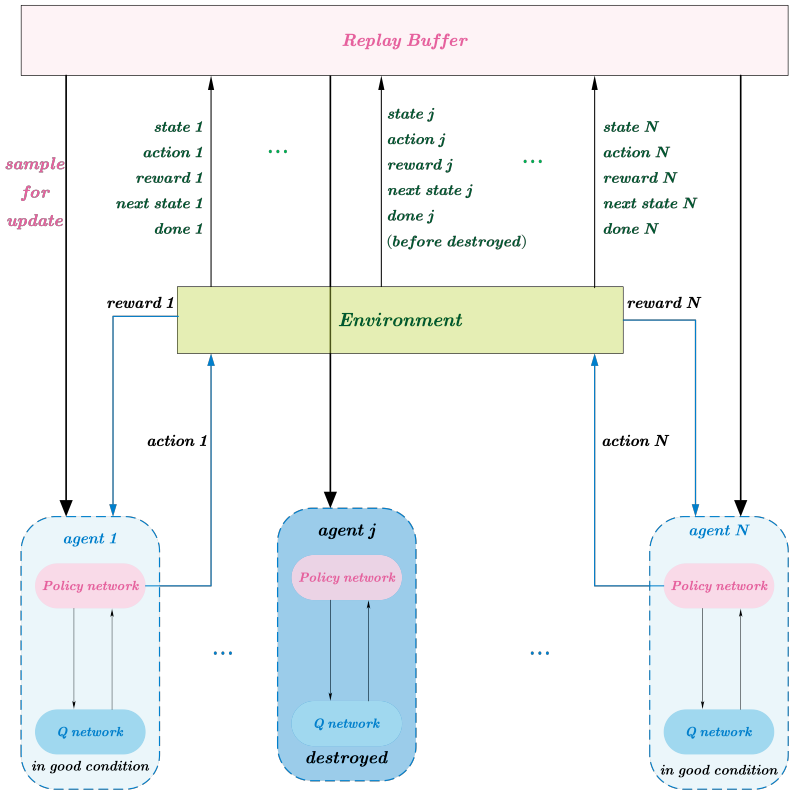

Figure 3. Overview of FA-MADDPG approach.

3.2.1. Details for FA-MADDPG

The application of the FA-MADDPG method in training will cause two problems, and we give the following solutions. Problem 1: After some agents crash, it will also have an impact on the decisions of other normal agents, which cannot be ignored. To solve this problem, we can set the observation of the other agents to a special value. In this scenario, the observation of the agent is set to the positions of all agents relative to the target in the environment. The observation value can be set to a special constant value in other normal agents so that the other normal agents can obtain information about whether the agent crashed or not. Problem 2: If the crashed agent does not interact with the environment, then the environment does not generate actions, states, or rewards, resulting in dimensional changes in the data input to the NNs. For this problem, when updating the NNs, the data generated by the interaction between the remaining agents and the environment after the agent's crash is not used. However, this does not mean that the data generated by the rest of the normal agents interacting with the environment after an agent crash are meaningless. This is because the rewards given by the environment to the agents are important measures of algorithm performance. We still needed to use this part of the data; that is to say, if agents crashed during training, then we divided the data generated by the interaction between all agents and the environment in this episode into two parts. The experience pool stored only

the data before the agent crashed, and we calculated the total reward during training using data from the entire episode. The results in Section 4 prove that this treatment is indeed effective. The whole algorithm flow is shown in Algorithm 1.

Algorithm 1: Frozen-Agent Method for MADDPG.

for *episode*=1 ***to*** M **do**
  Initialize a random process $\mathcal{N}$ for action exploration;
  Receive initial state $\mathbf{x}$;
  for t=1 **to** *max-episode-length* **do**
    For each agent i , select action $a_i = \mu_{\theta_i}(o_i) + \mathcal{N}_i$ for current policy and exploration;
    Execute actions $a = (a_1, a_2, \ldots, a_N)$ and observe reward r and new state $\mathbf{x}$
    if *no agent is crashed* **then**
      Store $(\mathbf{x}, a, r, \mathbf{x}')$ into replay buffer $\mathcal{D}$
    else Store into replay buffer $(\mathbf{x}, a, r, \mathbf{x}')$ for the agents not crashed and store a specific value into replay buffer $\mathcal{D}$ for the agents crashed;
    $\mathbf{x} \leftarrow \mathbf{x}'$;
    for *agent*=1 ***to*** N **do**
      Sample a random minibatch of L samples $\left(\mathbf{x}^j, a^j, r^j, \mathbf{x}'^j\right)$ from $\mathcal{D}$ and ensure there is no specific value;
      Set $y^j = r_i^j + \gamma Q_i^{\mu'}\left(\mathbf{x}'^j, a'_1, \ldots, a'_N\right)\Big|_{a'_k = \mu'_k\left(o_k^j\right)}$;
      Update Q by minimizing the loss
      $\mathcal{L}(\theta_i) = \frac{1}{L}\sum_j \left(y^j - Q_i^{\mu}\left(\mathbf{x}^j, a_1^j, \ldots, a_N^j\right)\right)^2$;
      Update policy by using the sampled policy gradient
      $\nabla_{\theta_i} J \approx \frac{1}{L}\sum_j \nabla_{\theta_i}\mu_i\left(o_i^j\right)\nabla_{a_i} Q_i^{\mu}\left(\mathbf{x}^j, a_1^j, \ldots, a_i, \ldots, a_N^j\right)\Big|_{a_i=\mu_i\left(o_i^j\right)}$;
    Update policy by using the sampled policy gradient.

3.2.2. Reward Function and Observation Settings

We set the observation value of each agent to the positions of the other agents relative to the target. If other agents crashed, then the position of the crashed agent relative to the target was recorded by the observation of the current agent as $(-1, -1)$.

The setting of the reward function is an effective incentive for the agent in the training process. Therefore, a reasonable set of rewards is crucial to the training process. The goal of the attacking agent is to hit the target while staying as far away from the defending agent as possible to avoid crashing. If only the collision reward is set, then a sparse reward problem will be caused. We present shaped reward functions. The reward function of the ith attacking agent consists of two parts in the following forms:

$$\begin{aligned} r_i &= r_i^{dis} + r_i^{col} \\ r_i^{dis} &= -d(i, goal) + \min_j d\left(i, good_j\right) \\ r_i^{col} &= \begin{cases} +5, if\ i\ hits\ goal \\ -5, if\ i\ collides\ with\ any\ defending\ agent \end{cases} \end{aligned} \tag{5}$$

where r_i^{dis} is the distance reward and r_i^{col} is the collision reward. $d(i, goal)$ refers to the distance between the attacking agent i and the target. $d\left(i, good_j\right)$ refers to the distance between the attacking agent and defending agent j .

The goal of the defending agents is to prevent the attacking agents from hitting the target and avoid colliding with the attacking agents. The reward function for the defending

agents was divided into two parts. One part was r_i^{att}, obtained according to the distance between the target and the attacking agents. The other part was r_i^{def}, based on the distance between the defending agents and the attacking agents. The reward function of the ith defending agent has the following form:

$$\begin{aligned} r_i &= r_i^{att} + r_i^{def} \\ r_i^{att} &= \begin{cases} \min\limits_j d(goal, att_j) - 5, if\ att_j\ hits\ goal \\ \min\limits_j d(goal, att_j), if\ att_j\ doesn't\ hit\ goal \end{cases} \\ r_i^{def} &= \begin{cases} -\sum\limits_j d(i, att_j) - 1, if\ i\ hits\ att_j \\ -\sum\limits_j d(i, att_j), if\ i\ doesn't\ hit\ att_j \end{cases} \end{aligned} \tag{6}$$

where $d(goal, att_j)$ is the distance between the attacking agent j and the target and $d(i, att_j)$ is the distance between the defending agent i and the attacking agent j. As we can see, both r_i^{att} and r_i^{good} are made up of a distance reward and collision reward. If the attacking agent collides with the target, then a -5 collision reward is given to the defending agents, and the training of this episode ends. If the defending agent collides with the attacking agent, then the defending agent is given a collision reward of -1. Through the setting of such a reward function, the defending agents can approach the attacking agents without impacting the agent.

In this scenario, it is assumed that defending agents work together to defend against attacking agents. We set up a cooperative reward function for the defending agents so that they could learn to cooperate. The reward of each defending agent is the same and is the average value of the reward for all defending agents:

$$r_{coop} = \frac{1}{n} \sum_{i=1}^{n} r_i \tag{7}$$

As can be seen from the setting of the reward function above, this is a complex game involving cooperation and competition.

4. Training and Execution Results

In this section, we will show how the FA-MADDPG method is trained and executed and how it works in practice.

4.1. Experiment Settings

To demonstrate the advantages of the FA-MADDPG algorithm, in this section, we changed the number of agents on both sides of the confrontation and conducted multiple simulations to verify the performance of this algorithm and its robustness to the changes in agent number.

In our experiments, the hyperparameters were set as follows. We use the Adam optimizer with a learning rate of $\tau = 0.01$ for soft updating the policy networks and Q networks. The discount factor γ was set to 0.95. The size of the replay buffer was 1×10^6, and we updated the network parameters after every 100 samples added to the replay buffer. The batch size for updating was 1024. The configurations of the neural networks are shown in Table 1.

Table 1. NN configurations.

Parameter	Value
Policy network hidden layers	2
Policy network hidden units	64
Q network hidden layers	2
Q network hidden units	64
Activation function	ReLU

The experiment was carried out on a computer with an Intel Core i5-12400CPU, 16 GB of RAM, and an Nvidia GTX 2060 GPU.

4.2. Experiment Results and Analysis

In the multi-agent ODG, both opposing parties want to achieve their goals. In this paper, the DRL method was applied to obtain the strategy that maximized each agent's own return, rather than the strategy of victory or dominance of one party. Since the crash reward was not calculated during training, comparing the total reward of all intelligent bodies cannot reflect the performance of this algorithm well. Therefore, we evaluated the performance of the algorithm by comparing the average reward of all agents.

To fit in with the actual confrontation scenario, we randomly generated the initial positions of each agent in the canvas at the beginning of each episode. To verify the robustness of the algorithm proposed in this paper, we ran experiments in four scenarios with different numbers of agents on both sides. The number of agents in four scenarios is shown in Table 2.

Table 2. Number of agents in different scenarios.

Scenario Number	Number of Defending Agents	Number of Attacking Agents
1	3	3
2	3	4
3	3	5
4	4	3

4.2.1. Reward Curves during Training

In four different scenarios, we applied the FA-MADDPG and MADDPG algorithms to train 8000 episodes. In Figure 4, the horizontal axis is the number of trained episodes, and the vertical axis is the average reward for all normal agents in that episode. In the MADDPG algorithm training process, we assumed that the agent still ran normally after a collision. As can be seen from the training results, in each scenario, the reward curve of the FA-MADDPG algorithm is higher than that of MADDPG algorithm, and the fluctuation range of the reward curve is smaller.

4.2.2. Execution Results

When we used the MADDPG algorithm for training, we had to ignore the situation of agent crashes during training and consider the situation in the execution stage. In this module, we illustrated the performance and robustness of the FA-MADDPG algorithm by comparing the execution effects of the FA-MADDPG and MADDPG algorithms in several different scenarios. In each scenario, the initial position of each agent was randomly generated, and 50,000 iterations were carried out. Assume that the task completion condition is that any attacking agent hits the target or all attacking agents are intercepted by defending agents. In each episode, the completion of the task or the number of iteration steps reaches a threshold that is considered the end of the execution episode. In Figure 5, a slice of the execution process is shown. When the agents were destroyed, they no longer interacted with the environment. They are expressed as gray circles in the figure. In fact, they no longer existed after crashing. The other agents worked as usual and did not consider the gray circles in the figure. The execution rewards for the four scenarios are shown in Figure 6. In the simulation stage, we will illustrate the superior performance of the FA-MADDPG algorithm from three aspects: the execution reward, the number of episodes executed, and the task completion rate.

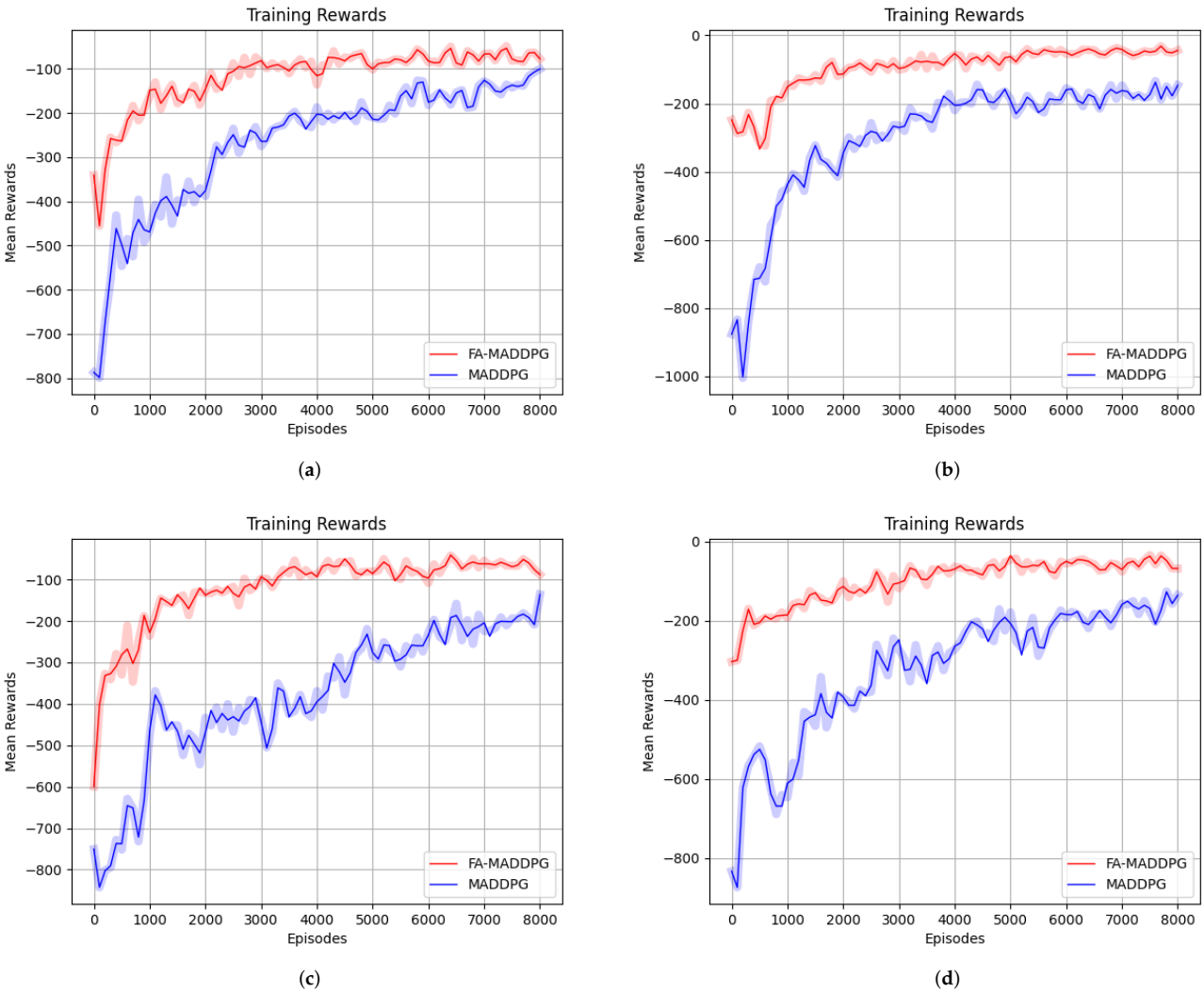

Figure 4. Training reward curves in different scenarios: (**a**) 3 vs. 3 training reward curves, (**b**) 3 vs. 4 training reward curves, (**c**) 3 vs. 5 training reward curves, and (**d**) 4 vs. 3 training reward curves.

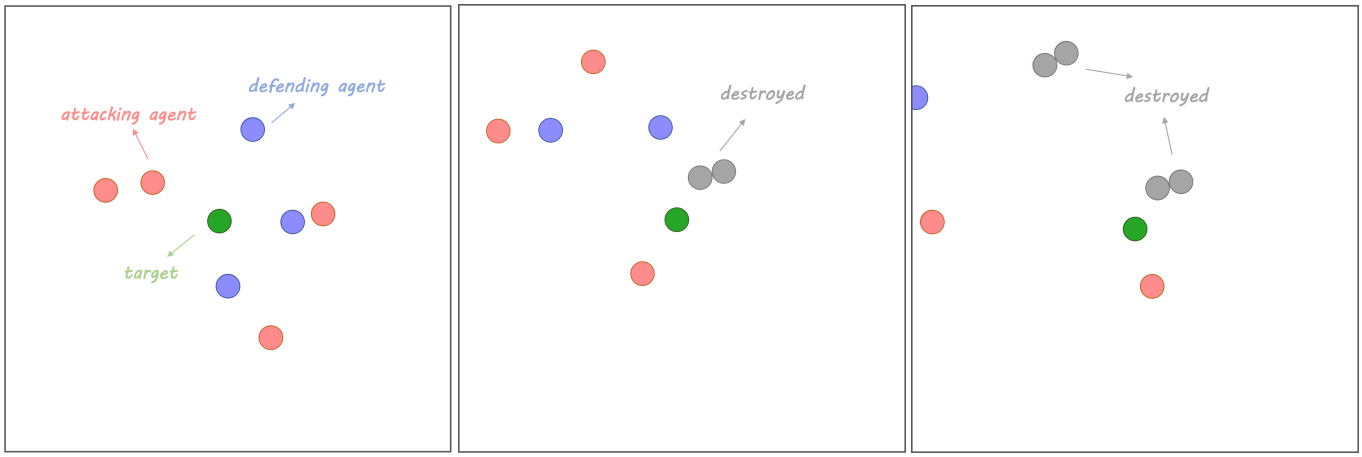

Figure 5. Render slices in Scenario 4.

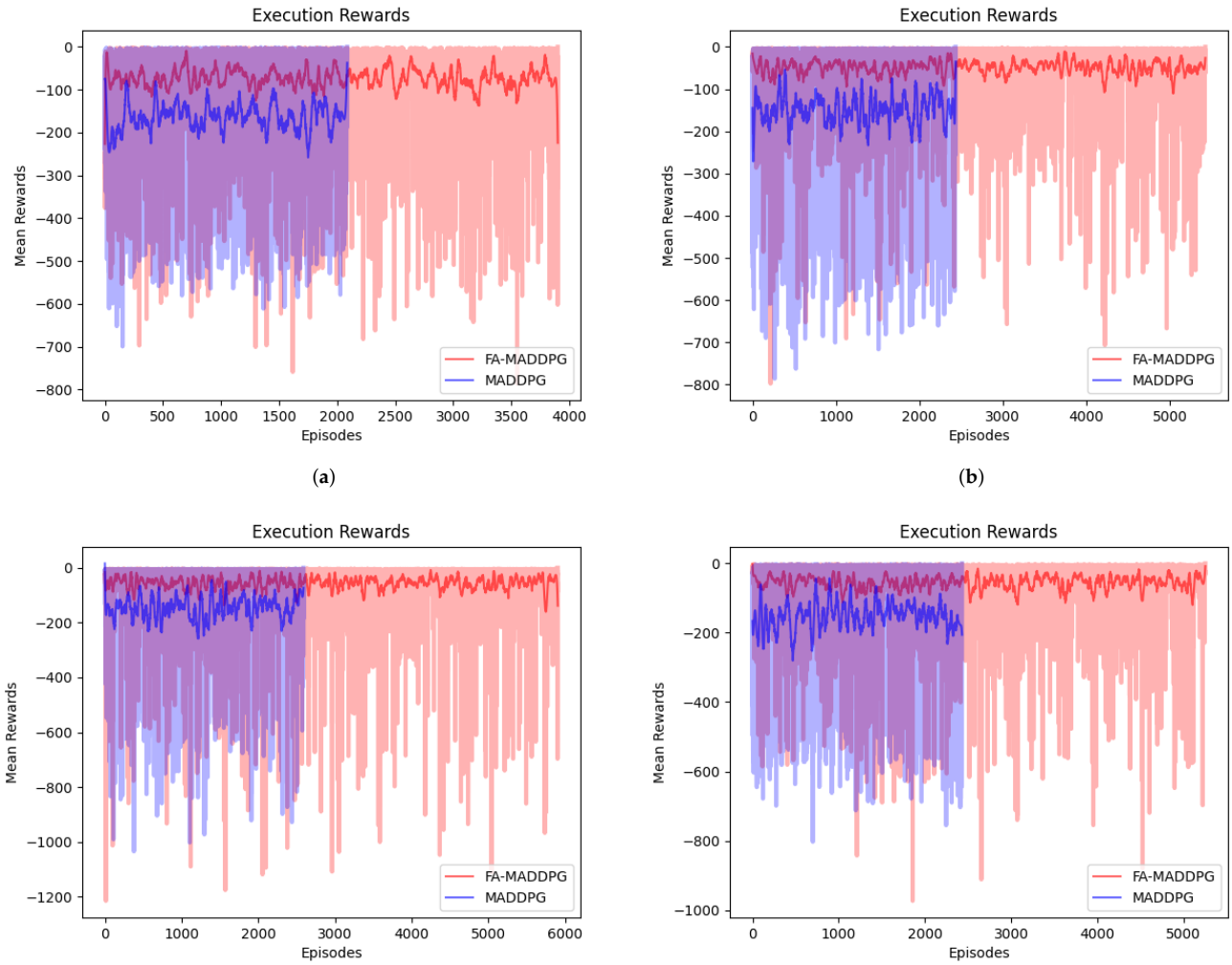

Figure 6. Execution reward curves in different scenarios: (**a**) 3 vs. 3 execution reward curves, (**b**) 3 vs. 4 execution reward curves, (**c**) 3 vs. 5 execution reward curves, and (**d**) 4 vs. 3 execution reward curves.

The number of executed episodes and average rewards of the two algorithms in each scenario are shown in Table 3. In the following three aspects, the performance of the FA-MADDPG algorithm was better than that of the MADDPG algorithm: (1) Execution rewards, where the reward values of the FA-MADDPG algorithm were significantly higher than those of the MADDPG algorithm; in other words, agents could make decisions that gave them greater returns with the FA-MADDPG algorithm; (2) the number of episodes executed, as with the same number of iterations, the number of episodes executed by the MADDPG algorithm was almost 50% that of the FA-MADDPG algorithm; and (3) task completion rate. Figure 7 shows the task completion rates of the two algorithms. Compared with the MADDPG algorithm, the FA-MADDPG algorithm had a higher action efficiency and task completion rate, and the average completion rate in the four scenarios was nearly 37% higher than for the MADDPG algorithm. All of this verifies the superior performance of the FA-MADDPG algorithm. For other algorithms such as DDPG, we also needed to assume that no agents crashed during training, so we just compared them with the MADDPG algorithm. It is proven that the MADDPG algorithm usually performs better than other algorithms in the multi-agent scenarios [15,19].

Table 3. Execution episodes and rewards for two algorithms.

Scenario Number	Algorithm	Execution Episodes	Average Reward
1	FA-MADDPG	3902	−72.47
	MADDPG	2088	−163.27
2	FA-MADDPG	5428	−48.48
	MADDPG	2435	−148.8
3	FA-MADDPG	5906	−55.23
	MADDPG	2591	−145.15
4	FA-MADDPG	5254	−54.83
	MADDPG	2432	−151.25

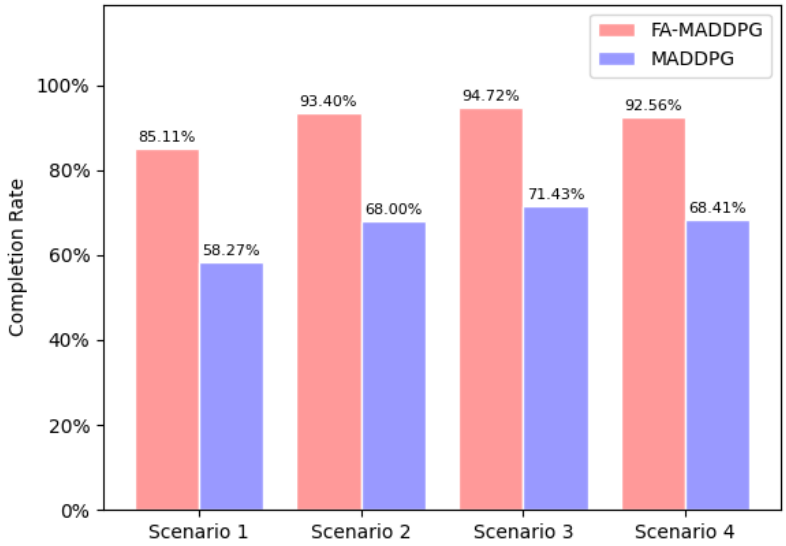

Figure 7. Comparison of the completion rates.

5. Conclusions

This paper mainly studied the problem of a multi-agent ODG based on the improved MADDPG algorithm and solved the problem of the MADDPG algorithm not coping with the reduction in the number of agents in the training process. First, we built a multi-agent SMG model. The FA-MADDPG algorithm was proposed to fix the defect that the MADDPG algorithm could not deal with the reduction in the number of agents due to collisions when training the model. Under the premise of not changing the structure of the NN, the agent crashed due to collisions during training and not interacting with the environment, and the NN parameters were updated with specific data. In order to solve the sparse reward problem, we designed the distance–collision reward function. Finally, we conducted training and simulation experiments under different scenarios with four groups of agents. The FA-MADDPG algorithm can successfully deal with a reduction in the number of agents in multi-agent scenarios. Aside from that, the results from three aspects showed that the FA-MADDPG algorithm has excellent performance. The rewards of the FA-MADDPG algorithm were significantly higher than those of the MADDPG algorithm. The number of episodes executed by the MADDPG algorithm was almost 50% that of the FA-MADDPG algorithm under the same iterations. The average completion rate in the four scenarios was nearly 37% higher than that of the MADDPG algorithm.

The multi-agent ODG is a very complicated game problem. When applying the MADDPG method to solve this kind of problem, there are still many difficulties, such as the number of agents on both sides increasing due to reinforcement, the agents being heterogeneous, and large-scale game confrontation. These will be the areas we need to focus on in the future. In addition, the application of this algorithm to the actual drone swarm or ground robot confrontation will also be one of the focuses of our future works.

Author Contributions: Conceptualization, X.L. and Z.L.; methodology, X.L.; software, X.L.; validation, Z.L., X.Z., X.Y. (Xuebo Yang) and X.Y. (Xinghu Yu); formal analysis, X.L.; investigation, Z.L.; resources, X.L.; data curation, X.L.; writing—original draft preparation, X.L.; writing—review and editing, Z.L.; visualization, X.L.; supervision, Z.L.; project administration, Z.L., X.Y. (Xuebo Yang) and X.Y. (Xinghu Yu); funding acquisition, Z.L., X.Z., X.Y. (Xuebo Yang) and X.Y. (Xinghu Yu). All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Natural Science Foundation of China, grant numbers 62273122 and U21B6001.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data are not publicly available due to privacy.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Chen, J.; Zha, W.; Peng, Z.; Gu, D. Multi-player pursuit–evasion games with one superior evader. *Automatica* **2016**, *71*, 24–32. [CrossRef]
2. Margellos, K.; Lygeros, J. Hamilton–Jacobi Formulation for Reach–Avoid Differential Games. *IEEE Trans. Autom. Control.* **2011**, *56*, 1849–1861. [CrossRef]
3. Zhou, Z.; Zhang, W.; Ding, J.; Huang, H.; Stipanović, D.M.; Tomlin, C.J. Cooperative pursuit with Voronoi partitions. *Automatica* **2016**, *72*, 64–72. [CrossRef]
4. Chen, M.; Zhou, Z.; Tomlin, C.J. Multiplayer reach-avoid games via pairwise outcomes. *IEEE Trans. Autom. Control.* **2016**, *62*, 1451–1457. [CrossRef]
5. Zou, B.; Peng, X. A Bilateral Cooperative Strategy for Swarm Escort under the Attack of Aggressive Swarms. *Electronics* **2022**, *11*, 3643. [CrossRef]
6. Zhang, S.; Ran, W.; Liu, G.; Li, Y.; Xu, Y. A Multi-Agent-Based Defense System Design for Multiple Unmanned Surface Vehicles. *Electronics* **2022**, *11*, 2797. [CrossRef]
7. Yang, K.; Dong, W.; Cai, M.; Jia, S.; Liu, R. UCAV Air Combat Maneuver Decisions Based on a Proximal Policy Optimization Algorithm with Situation Reward Shaping. *Electronics* **2022**, *11*, 2602. [CrossRef]
8. Zhao, X.; Yang, R.; Zhang, Y.; Yan, M.; Yue, L. Deep Reinforcement Learning for Intelligent Dual-UAV Reconnaissance Mission Planning. *Electronics* **2022**, *11*, 2031. [CrossRef]
9. Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; Bengio, Y. Generative adversarial networks. *Commun. ACM* **2020**, *63*, 139–144. [CrossRef]
10. Ibrahim, A.M.; Yau, K.-L.A.; Chong, Y.-W.; Wu, C. Applications of Multi-Agent Deep Reinforcement Learning: Models and Algorithms. *Appl. Sci.* **2021**, *11*, 10870. [CrossRef]
11. Qi, H.; Huang, H.; Hu, Z.; Wen, X.; Lu, Z. On-Demand Channel Bonding in Heterogeneous WLANs: A Multi-Agent Deep Reinforcement Learning Approach. *Sensors* **2020**, *20*, 2789. [CrossRef]
12. Jung, S.; Yun, W.J.; Kim, J.; Kim, J.-H. Coordinated Multi-Agent Deep Reinforcement Learning for Energy-Aware UAV-Based Big-Data Platforms. *Electronics* **2021**, *10*, 543. [CrossRef]
13. Chen, C.; Ma, F.; Xu, X.; Chen, Y.; Wang, J. A Novel Ship Collision Avoidance Awareness Approach for Cooperating Ships Using Multi-Agent Deep Reinforcement Learning. *J. Mar. Sci. Eng.* **2021**, *9*, 1056. [CrossRef]
14. Liang, L.;Deng, F.; Peng, Z.; Li, Xi.; Zha, W. A differential game for cooperative target defense. *Automatica* **2019**, *102*, 58–71. [CrossRef]
15. Lowe, R.; Wu, Y.I.; Tamar, A.; Harb, J.; Pieter A.; Mordatch, I. Multi-agent actor-critic for mixed cooperative-competitive environments. *Adv. Neural Inf. Process. Syst.* **2017**, *30*, 6382–6393.
16. Wan, K.; Wu, D.; Zhai, Y.; Li, B.; Gao, X.; Hu, Z. An Improved Approach towards Multi-Agent Pursuit–Evasion Game Decision-Making Using Deep Reinforcement Learning. *Entropy* **2021**, *23*, 1433. [CrossRef] [PubMed]
17. Xiang, L.; Xie, T. Research on UAV Swarm Confrontation Task Based on MADDPG Algorithm. In Proceedings of the 2020 5th International Conference on Mechanical, Control and Computer Engineering (ICMCCE), Harbin, China, 25–27 December 2020.
18. Li, P.; Jia, S.; Cai, Z. Research on Multi-robot Path Planning Method Based on Improved MADDPG Algorithm. In Proceedings of the 2021 China Automation Congress (CAC), Beijing, China, 22–24 October 2021.
19. Zhang, R.; Zong, Q.; Zhang, X.; Dou, L.; Tian, B. Game of Drones: Multi-UAV Pursuit-Evasion Game With Online Motion Planning by Deep Reinforcement Learning. *IEEE Trans. Neural Netw. Learn. Syst.* **2022**. [CrossRef] [PubMed]
20. Littman, M.L. Markov games as a framework for multi-agent reinforcement learning. In *Machine Learning Proceedings 1994*; Elsevier: Amsterdam, The Netherlands, 1994; pp. 157–163.

21. Shao, K.; Zhu, Y.; Zhao, D. StarCraft Micromanagement With Reinforcement Learning and Curriculum Transfer Learning. *IEEE Trans. Emerg. Top. Comput. Intell.* **2019**, *3*, 73–84. [CrossRef]
22. Peng, P.; Wen, Y.; Yang, Y.; Yuan, Q.; Tang, Z.; Long, H.; Wang, J. Multiagent bidirectionally-coordinated nets: Emergence of human-level coordination in learning to play starcraft combat games. *arXiv* **2017**, arXiv:1703.10069.