Reinforcement learning of cluster robots based on MADDPG algorithm

Yifei Chen
12010502

Jiajun Long
12011624

Yijun Fang
12011301

***Abstract*—Reinforcement learning of multiple intelligences is very important and effective for cluster-based robot development. Based on OpenAi's multiagent-particle-envs environment, we implement the multi-intelligent body reinforcement learning MADDPG algorithm. By studying the algorithm of the paper and implementing it using pytorch according to our understanding. We also modified the critic-actor neural network. The algorithmic part of the MADDPG that we have built is able to interface with the experimental scenarios that we have built ourselves, enabling the learning of scenarios such as cooperation/competition/communication between robots. Finally, we analyze and summarize the learning effects of clustered robots performing different tasks.**

***Index Terms*—MADDPG, critic-actor network, cluster robots, multi-intelligent**

I. Introduction

Clustered robots extract engineering principles from the study of these natural systems to build multi-robot systems with comparable capabilities. In this way, cluster robots aim to build systems that are more robust, more fault-tolerant, and more flexible than individual robots, and that can better adapt their behavior to environmental changes. Implementing swarm behavior in robots requires more than applying swarm intelligence algorithms to existing robotic platforms. In fact, researchers need to completely rethink traditional robot functions such as perception, control, localization, and the design of the robot platform itself. Over the past two decades, researchers in cluster robotics have made significant progress, providing proof-of-concept for the potential of cluster robots and enabling researchers to better understand how complex behaviors emerge in nature. Nonetheless, translating this research into practice remains fraught with challenges that need to be properly addressed by researchers. Indeed, to date, only a few experiments have successfully demonstrated a large number of autonomous self-organizing robots, and there is still a gap in practical applications of clustered robots.

Research on complex behaviors of clustered robots using reinforcement deep learning is also popular nowadays. Different terrain obstacle conditions are encountered during the training process, and the optimal convergence solution is eventually found. At the same time, the grappler is trained simultaneously in this process as a dynamic factor in the system,which makes the training environment of the grappler more complex and observes its training effect. It is obtained from the way of evolutionary selection in nature.

The traditional reinforcement learning algorithms that we are most often exposed to are single-intelligent reinforcement learning algorithms, but there are also many important application scenarios that involve the interaction between multiple intelligences, such as the control of multiple robots, language communication, multi-player games, etc. [2], and the clustered robots mentioned in this paper. Open-AI's MADDPG (Multi-Agent Policy Gradient) algorithm is able to train and test multiple intelligences in a stable and efficient way [3].

II. Expiation of the algorithm

A. Reinforcement Learning Introduction

The basic concept of reinforcement learning, we believe, can be divided into 2 layers.

Layer 1: Reinforcement learning consists of two parts: the intelligences (AGENTS) and the environment (ENV). During the reinforcement learning process, the intelligent body and the environment are always interacting.

Layer 2: After an intelligence acquires a state in the environment, it uses that state to output an action, which is also called a decision. This action is executed in the environment, and the environment outputs the next state and the reward for the current action based on the action taken by the intelligence. The goal of the intelligence is to obtain as many rewards as possible from the environment.

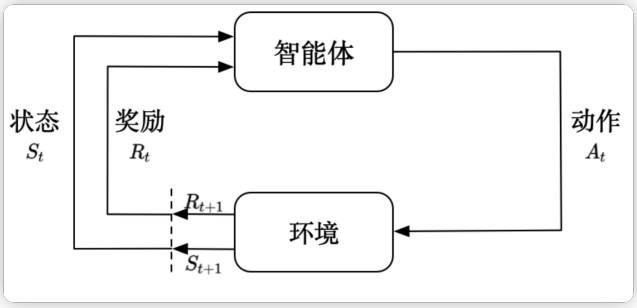

Fig. 1. The basic structure of reinforcement learning

We believe that the fastest way to cognitively reinforce learning is to use the analogy with supervised learning that we have learned. Suppose we train a classifier, such as a neural network. In order to distinguish whether the input image is a car or an airplane, the correct label information needs to be passed to the neural network during the training process. When

the neural network makes a wrong prediction, such as the input picture of a car, and it predicts that it is an airplane, we tell it directly that the prediction is wrong and the correct label should be car. Finally we write a loss function based on similar errors and train the neural network by back propagation. First of all the observation obtained by the intelligence is not independently and identically distributed, there is actually a very strong continuity between the previous frame and the next frame. The data we get are correlated time series data, which do not satisfy the independent identical distribution. Second, for each decision action, there is no label to tell the intelligence whether it is good or wrong. Only at the end of a round can the good or bad action taken be known by a REWARD. Finally, the process of acquiring an intelligence's capabilities is actually a trial-and-error exploration process. The intelligent body will be in the decision making time to explore the unknown way or use the existing experience value to make a judgment. We can see from these characteristics that the actions of the intelligent body affect the data it subsequently obtains. In the process of training an intelligent body, many times we also get data by the interaction of the intelligent body that is learning with the environment. So if the intelligences do not remain stable during the training process, it makes the data we collect very bad. We train the intelligence through the data, and if there is a problem with the data, the whole training process will fail. So a very important problem in reinforcement learning is how to keep the intelligent body's action steadily improving all the time. We can also think of stock trading as a reinforcement learning process. We can continuously buy and sell stocks and then learn how to buy and sell based on the feedback given by the market to maximize our reward.

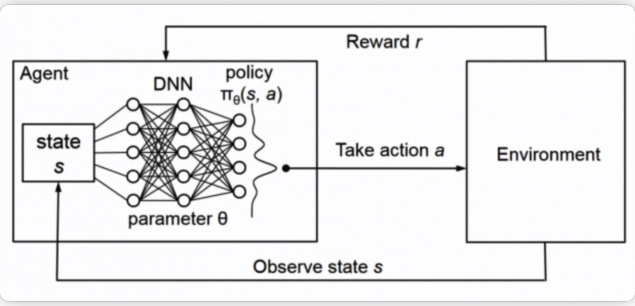

Fig. 2. The procedure of reinforcement learning

B. Influences of RL decision making

- **Policy:**The intelligence will use the strategy to pick the next action.
- **Value function:**We use the value function to evaluate the current state. The value function is used to evaluate how much of an effect the entry of an intelligence into a state can have on the rewards that follow. The larger the value function, the more favorable it is for the intelligence to enter the state.
- **Model:**The model represents the intelligence's understanding of the state of the environment, which determines how the world operates in the environment.

$$\nabla J(\theta) = E_{s\sim p^\pi, a\sim\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a \mid s) Q^\pi(s,a)\right] \quad (1)$$

$$L(\theta) = E_{s,a,r,s'}\left[\left(Q^*(s,a \mid \theta) - y\right)^2\right] \quad \text{with } y = r + \gamma \max_{a'} \bar{Q}^*(s', a') \quad (2)$$

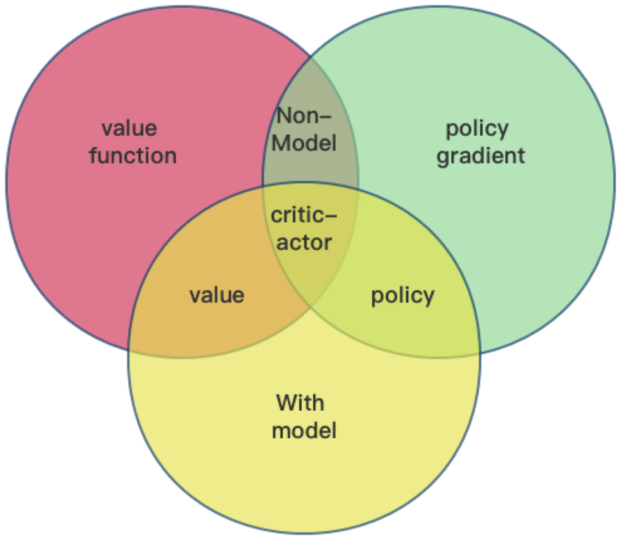

Fig. 3. The relationship between factors that influence decision making

The value-based agent learns the value function explicitly and its strategy implicitly. The policy is derived from its learned value function (the value function that follows the action that maximizes the value of this function). policy-based agent learns the policy directly (gets an optimal action in each state). Value-based reinforcement learning algorithms such as Q-learning, and policy-based reinforcement learning algorithms such as Policy Gradient (PG) algorithm are generally used in processes with large scale and continuous actions, such as robot movement and grasping.

Modeled reinforcement learning refers to building a virtual world based on the experience in the environment and learning in both the real environment and the virtual world; model-free reinforcement learning refers to learning the optimal policy without modeling the environment and interacting with the real environment directly. Currently, most deep reinforcement learning methods use model-free reinforcement learning because: model-free reinforcement learning is simpler, more intuitive, and has rich open source material, such as the AlphaGo series, which uses model-free reinforcement learning; in most of the current reinforcement learning research, the environment is static and describable, and the state of the intelligence is discrete and observable (e.g., Atari game platform), such relatively simple and deterministic problems do not require the evaluation of state transfer functions and

reward functions, and can be directly trained using model-free reinforcement learning with a large number of samples to obtain better results [4].

C. DDPG algorithm(Deep Deterministic Policy Gradient)

The strategy network plays the role of the actor, which is responsible for presenting the output to the outside world and outputting the action. q network is the commentator, which will evaluate the actor's output at each step, scoring it and estimating how much the actor's action will be rewarded in the future, i.e., estimating what the Q value of the actor's output is approximately, i.e., $Q_w(s,a)$ The actor needs to make an action based on the current state of the stage an action. The commentator is the judge, who needs to score the actor's performance based on the current state of the stage and the actor's output $Q_w(s,a)$ to adjust his strategy based on the judge's score, i.e., to update the actor's neural network parameters θ to do better next time. The reviewer, on the other hand, has to adjust his scoring strategy based on the feedback from the audience, i.e., the feedback reward from the environment, which means that he has to update the parameters of the reviewer's neural network. w The ultimate goal of the reviewer is to get the actor's performance to receive as much cheers and applause from the audience as possible, thus maximizing the total future gain.

At the very beginning of the training, the parameters of these two neural networks are randomized. So the reviewers are initially scored randomly and the actors output random actions. But since the reward of environmental feedback exists, the reviewers' scores become more and more accurate and the performance of the actors they judge becomes better and better. Since the actor is a neural network, a policy network that we want to train well, we need to compute gradients to update and optimize the parameters inside it θ. Simply put, we want to adjust the actor's network parameters so that the reviewer scores as high as possible. Note that the actor here is not concerned with the audience, it is only concerned with the judges, and it only caters to the judges' scores $Q_w(s,a)$

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

The optimal strategy for a deep Q network is to learn a good Q network, and after learning this network, we want to choose the action that maximizes the Q. The goal of DDPG is also to solve for the action that maximizes the Q value. The actor is just trying to satisfy the judges' scores, so the gradient of the optimization strategy network is to maximize the Q value, so the loss function is constructed so that Q takes a negative sign. When we write the code to put this loss function into the optimizer, it will automatically minimize the loss, that is, maximize Q.

In addition to the policy network to do the optimization, DDPG has a Q network to optimize as well. The reviewer does not know how to score at first, and it is learning step by step to give accurate scores slowly. The way we optimize

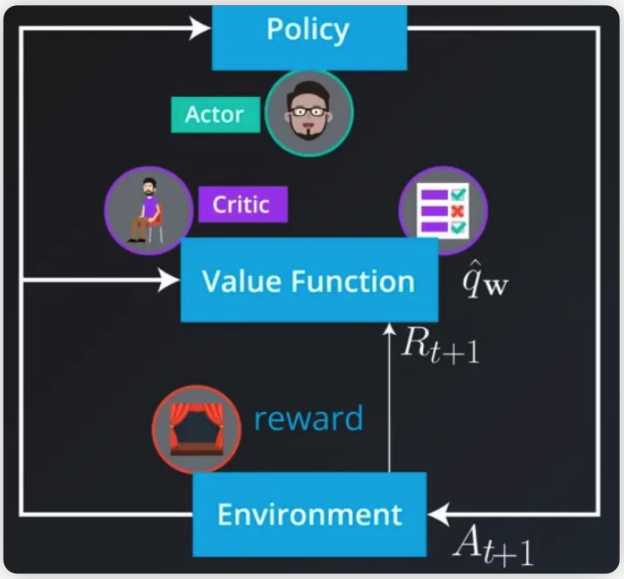

Fig. 4. The actor-critic agent neural network

the Q-network is actually the same as the deep Q-network The way we optimize the Q-network is the same, we fit the future reward Q_{target} with the real reward r and the next Q i.e. Q'. then let the output of the Q-network approximate Q_{target}. so the constructed loss function is the mean squared difference of these two values directly. After constructing the loss function, we put it into the optimizer and let it minimize the loss automatically.

When testing, we only need Actor to do it, and we don't need Critic's feedback at this point. Therefore, during training, we can add some additional information to the Critic stage to get a more accurate Q value, such as the state and actions of other intelligences, etc. This is what is meant by centralized training, i.e., each intelligence evaluates the value of the current action not only based on its own situation, but also based on the behavior of other intelligences.

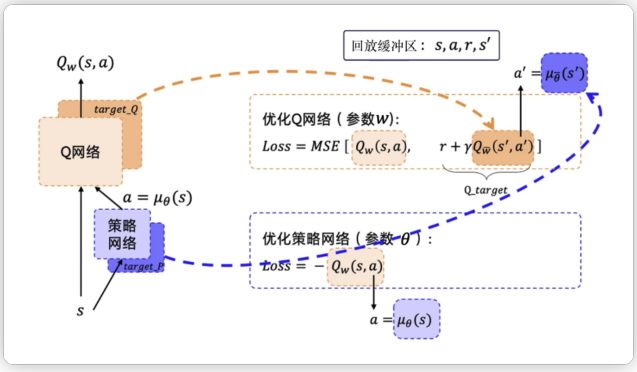

Fig. 5. DDPG with Qlearning and policy gradient

D. Introduction to MADDPG

Through the pseudo-code of MADDPG, we can see that MADDPG is similar to DDPG, but the main difference is

that in the DDPG algorithm, the input of Critic is its own observation-action data, while in MADDPG, the Critic of each agent not only inputs its own In MADDPG, the Critic of each Agent inputs not only his own observation and action information, but also other agents' action and observation (observation status) information, which means that the Critic of MADDPG can take score of the Actor from a more macroscopic perspective.

Algorithm 1: Multi-Agent Deep Deterministic Policy Gradient for N agents

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 $a_i = \boldsymbol{\mu}_{\theta_i}(o_i) + \mathcal{N}_t$ w.r.t. the current policy and exploration
    Execute actions $a = (a_1, \ldots, a_N)$ and observe reward r and new state $\mathbf{x}'$
    Store $(\mathbf{x}, a, r, \mathbf{x}')$ in replay buffer $\mathcal{D}$
    $\mathbf{x} \leftarrow \mathbf{x}'$
    for agent $i = 1$ to N **do**
      Sample a random minibatch of S samples $(\mathbf{x}^j, a^j, r^j, \mathbf{x}'^j)$ from $\mathcal{D}$
      Set $y^j = r_i^j + \gamma Q_i^{\boldsymbol{\mu}'}(\mathbf{x}'^j, a'_1, \ldots, a'_N)|_{a'_k = \boldsymbol{\mu}'_k(o_k^j)}$
      Update critic by minimizing the loss $\mathcal{L}(\theta_i) = \frac{1}{S}\sum_j \left(y^j - Q_i^{\boldsymbol{\mu}}(\mathbf{x}^j, a_1^j, \ldots, a_N^j)\right)^2$
      Update actor using the sampled policy gradient:

$$\nabla_{\theta_i} J \approx \frac{1}{S}\sum_j \nabla_{\theta_i} \boldsymbol{\mu}_i(o_i^j) \nabla_{a_i} Q_i^{\boldsymbol{\mu}}(\mathbf{x}^j, a_1^j, \ldots, a_i, \ldots, a_N^j)|_{a_i = \boldsymbol{\mu}_i(o_i^j)}$$

    end for
    Update target network parameters for each agent i:

$$\theta'_i \leftarrow \tau\theta_i + (1-\tau)\theta'_i$$

  end for
end for

Fig. 6. The pseudo-code of MADDPG

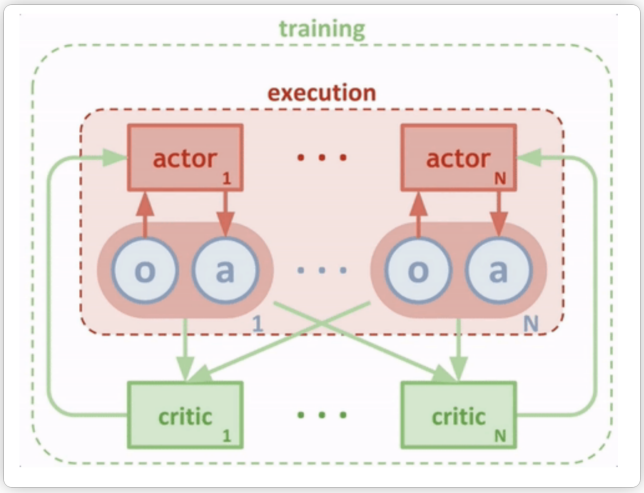

Fig. 7. The structure of MADDPG

III. Code structure

We first analysis the code of MADDPG.
For the training level of the M-set:
the number of neural networks in MADDPG:
If the number of intelligences is N
- The number of Actor networks is 2N

- Each intelligent body corresponds to an actor's training network and target network

- The number of Critic networks is 2N

- where each intelligentsia uses a separate critic training network and target network

1) The input X is the observation space of all our intelligences.
2) For each step of agent traversal is performed:
   for each agent, a policy function is used to derive our action, here we are using the neural network of the actor and a noise is added to it. This series is stored in the experience pool buffer.
3) for each agent:
   - Extract their information from the experience pool.
   - Update the critic's loss function function.
   - Update the actor loss function by policy gradient descent according to the feedback value of the critic.
4) Update the parameter values of the target neural network for optimization.

According to the analysis above, we construct our code as showed in fig.8. Its functions and roles in the entire projects is showed in the fig.9. We use a lot time to construct the buffer.py to store its learning information, in order to provide its best performance.

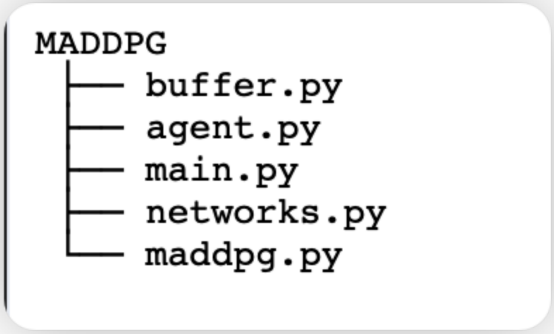

Fig. 8. The code structure of this project

IV. Training scenario Design

Multi-intelligent Particle environment is the OpenAi open source multi-intelligence learning environment built for the operation of the MADDPG algorithm proposed by OpenAi. For the convenience and ease of training, in this environment we can use the $make_env$ function to build the competition/collaboration/communication scenario we need including the number of agents, selecting the tasks they want to accomplish and the relationships (competition/collaboration/communication), such as cooperating to capture each other, maintaining distance, etc.. We can add different objects to them and set their different functions (forest, wall). State information can also be defined, mainly the coordinates/direction/speed of the smart body. The movement space of these balls can be continuous or discrete, and the discrete mode means that the movement of the smart body is discrete into several directions. In addition, in this environment, the collision between objects are able to simulate the actual collision of rigid bodies, by calculating the momentum,

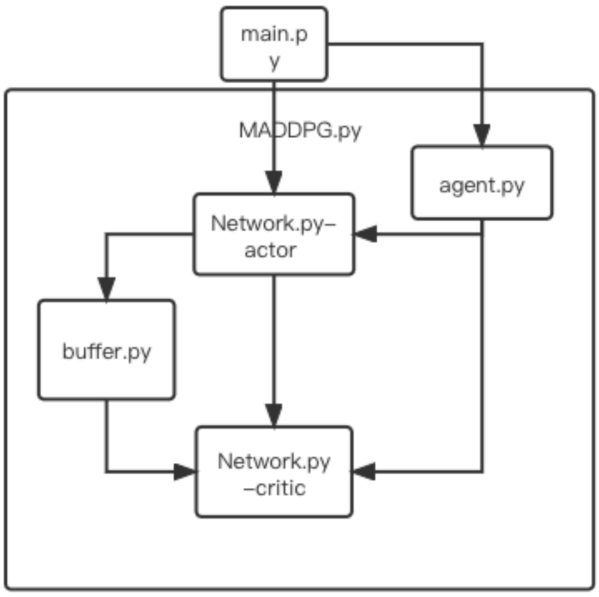

Fig. 9. The code function of this project

force, etc. to calculate the speed and displacement. To a certain extent, it is able to simulate the actual scene of our reality [6].

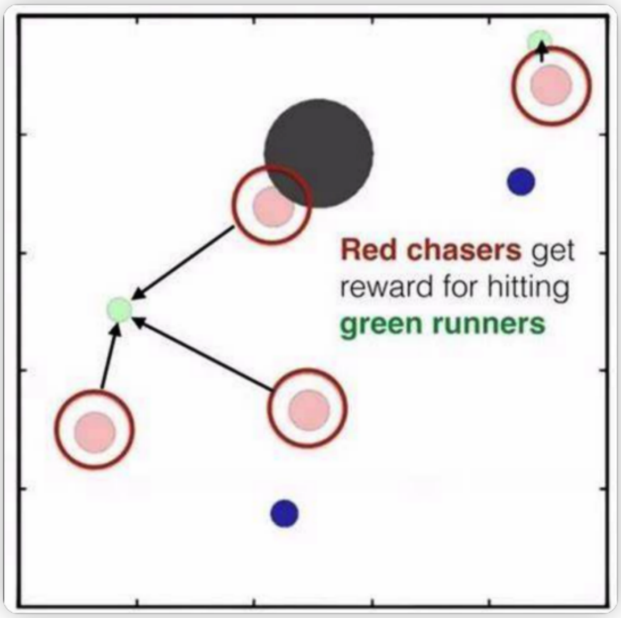

Fig. 10. The multi-particles environment from OpenAi

In order to simulate the motion and growth of clustered robots, we design two kinds of problems with different scenarios: Chasing and controlling. In the chasing problem, the fugitives are faster and wants to avoid being hit by the crowd of machines. The drones are set to a slower speed and chase down the intelligent escapees. There will be obstacles in them that will block their route. And there are also other features:

- There is food, and escapees are rewarded for being nearby.
- The bunker, when both sides are inside the bunker, is not seen by the outside world.
- The drone cluster has a ”leader” in the back of the training that can see where the drones are at all times and can communicate with other drones to coordinate the chase.

And different scenarios were designed to show different relationship between the cluster robots.

- A drone searches for a fixed point.
- A drone chases a fugitive.
- Multiple drones chasing a fugitive. Equal cooperation between each other.
- Multiple drones chasing multiple escapees. (equal relationship between each other, rewards for catching the most escapees).
- Multiple drones chasing multiple escapees. One of them is the leader drones and is able to combine the drones' location information for the chase (combined information).
- Two teams of drones chase the escapees. Each has a LEADER.

As for the controlling problem, The scenario consists of L landmarks, including a target landmark, N cooperative intelligences that know the target landmark and are rewarded according to their distance from the target, and M drones that must prevent the former from reaching the target. The drones do this by pushing each other away from the landmark and occupying it. Although the drones are also rewarded based on their distance from the target landmark, they do not know the correct target; this must be inferred from the actions of the opposing intelligences. Same as the chasing problem, we have different scenario for it.

- A drones tracking and control of a target. Distance control, keep away.
- Control of multiple drones against a target. Distance control, carry out a control and protection that can keep its relationship.
- Control of multiple drones on one target. There is a leader for the unification of information.

The specific motion performance can be seen in the video demos in the additional material.

V. Analysis of training performance

We have trained and simulated for the mentioned scenario. We found in our attempts that for the action space of the intelligence, the strategy he takes does not allow him to perform better convergence after a certain number of steps. This defined value we set to 25 steps. Here we look at the training performance for the two scenarios we set, where we set the number of training sessions at 10,000 for each scenario. The overall observation is that there will be better convergence properties around 5000-8000 steps. In fig.11 and fig.13, the relationship between them are equally, which is pretty simple, so their performance is relatively good.

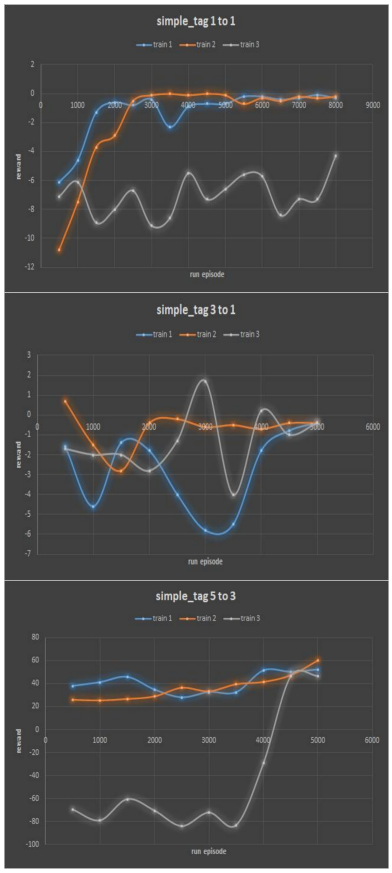

Fig. 11. The chasing problem of a)1 vs 1, b)3 vs 1, c)5 vs 3

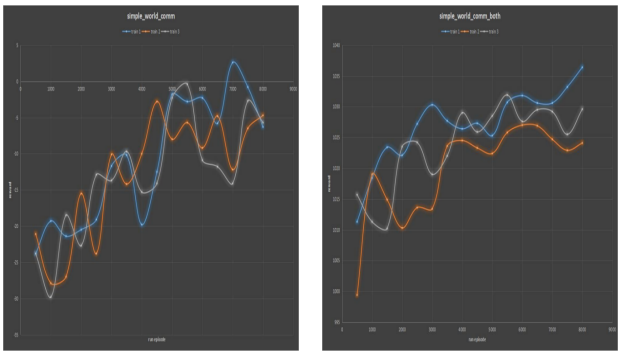

Fig. 12. Chasing problem with leaders

We first examine cooperative communication situations. In practice, we observed that listeners could learn to ignore the speaker and simply move to the middle of all observed landmarks. In fig.15 we plotted the learning curve over 2000 rounds. We read the literature and conjecture that the main reason for the failure of traditional RL methods in this (and other) multi-intelligence settings is the lack of consistent gradient signals. For example, the speaker is penalized if the correct signal is given when the speaker moves in the wrong direction. This problem becomes more severe as the time step increases: we observe that traditional strategic gradient methods can learn when the listener's goal is simply to reconstruct the observer in

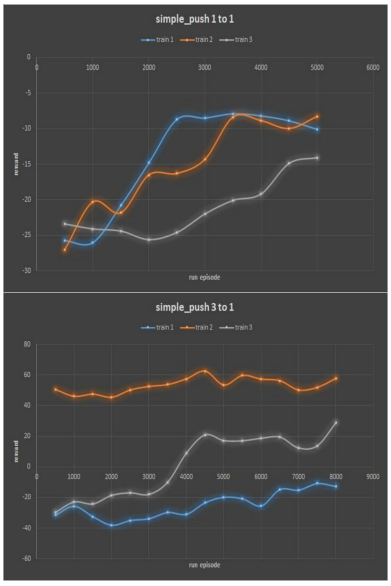

Fig. 13. The chasing problem of a)1 vs 1, b)3 vs 1.

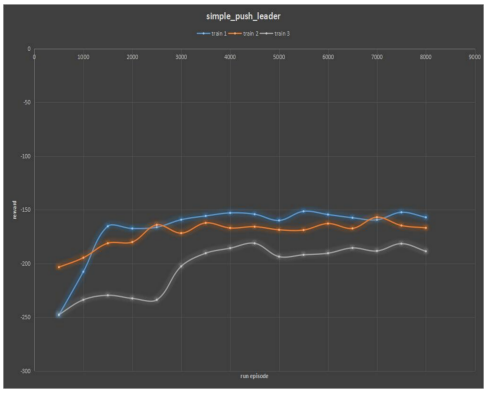

Fig. 14. Control problem with leaders

a single time step or whether the intelligences and landmarks are fixed and uniformly distributed. This suggests that many previously proposed multi-intelligence approaches for short time range scenes (e.g., [10]) may not be applicable to more complex tasks [1].

As for the scenarios with leaders, there performances is not good as the simple one. However, most of the cases can drop to a convergence in the end. From the simulation, we can see that the drones are already know the best policy to get the target in the most efficiency way, which you can see in the video demo in the support material. All in all, in terms of performance, the strategies used by the drone swarms with leaders were more effective than the drone swarms with equal relationships.

VI. CONCLUSION

The multiagent-particle-envs we used in this project is a simple environment that can only simulate and perform some simple tasks, but in the foreseeable future, human-created machines and systems will become more and more complex and capable, and in many tasks can reach the level of human

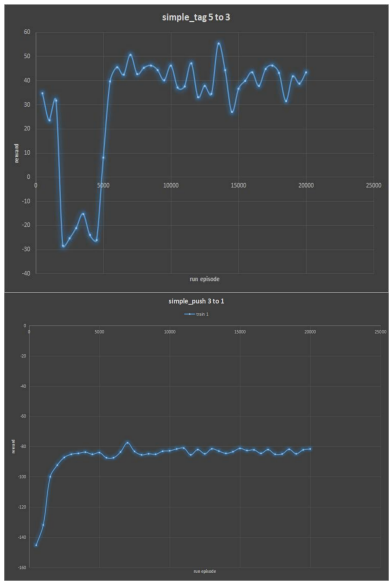

Fig. 15. Long training steps and time for two scenarios a)Tag 5 vs 3, b)Push 3 vs 1.

experts or even surpass them. Some people say that a machine is not conscious, it cannot think, it is just there mechanically executing algorithms without knowing the meaning of running them, and that the machine is not replicating human intelligence. But if we look at it from a pragmatic point of view, it doesn't matter whether the machine is conscious or not, whether it copies human "intelligence" or not, as long as the machine can do the task as required, just like a flying machine can fly because it is aerodynamic, and it is not necessary to understand how birds There is no need to understand how birds wave their wings and how their feathers work.

Drawing inspiration from the behavior of swarming insects or swarming species is valuable in many cases because the properties and behaviors exhibited by these natural clusters are the basis for arbitrarily intelligent robotic clusters: they are living evidence of the fact that self-organization can work universally, and they provide viable solutions to specific problems, such as how robotic clusters move in a coordinated fashion, distribute tasks, or make collective decisions. But our interpretation and understanding of them is also the hard part of our optimization, that is, how to focus for the tuning of our policy functions. Our direct control of the cluster is complex because understanding what the cluster is doing is very challenging due to the large amount of interaction that occurs within the cluster, which can be difficult to read for a human observer. Therefore, interpretability is critical. A possible solution might be built into the self-organizing mechanism of the cluster to allow the user to see the current state and goals of the cluster.

ACKNOWLEDGMENT

This project is completed by three members equally.

Yifei Chen is in charge of overall algorithm framework and research idea construction.

Jiajun Long is in charge of implementation of experimental simulation scenarios and visualization.

Yijun Fang is in charge of Specific implementation of the algorithm and training of the network.

REFERENCES

[1] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, Igor Mordatch, arXiv:1706.02275 [cs.LG] https://doi.org/10.48550/arXiv.1706.02275

[2] DeepMind AI reduces google data centre cooling bill by 40. https://deepmind.com/blog/deepmind-ai-reduces-google-data-centre-cooling-bill-40/. Accessed: 2017-05-19.

[3] M. Abadi and D. G. Andersen. Learning to protect communications with adversarial neural cryptography. arXiv preprint arXiv:1610.06918, 2016.

[4] C. Boutilier. Learning conventions in multiagent stochastic domains using likelihood estimates. In Proceedings of the Twelfth international conference on Uncertainty in artificial intelligence, pages 106–114. Morgan Kaufmann Publishers Inc., 1996.

[5] L. Busoniu, R. Babuska, and B. De Schutter. A comprehensive survey of multiagent reinforce- ment learning. IEEE Transactions on Systems Man and Cybernetics Part C Applications and Reviews, 38(2):156, 2008.

[6] G. Chalkiadakis and C. Boutilier. Coordination in multiagent reinforcement learning: a bayesian approach. In Proceedings of the second international joint conference on Autonomous agents and multiagent systems, pages 709–716. ACM, 2003.

[7] P. Dayan and G. E. Hinton. Feudal reinforcement learning. In Advances in neural information processing systems, pages 271–271. Morgan Kaufmann Publishers, 1993.

[8] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson. Counterfactual multi-agent policy gradients. arXiv preprint arXiv:1705.08926, 2017.

[9] J. N. Foerster, Y. M. Assael, N. de Freitas, and S. Whiteson. Learning to communicate with deep multi-agent reinforcement learning. CoRR, abs/1605.06676, 2016.

[10] A. Lazaridou, A. Peysakhovich, and M. Baroni. Multi-agent cooperation and the emergence of (natural) languag[]e. arXiv preprint arXiv:1612.07182, 2016.