Reinforcement Learning for a Multi-Agent System

Griffin Van Anne
University of Colorado Boulder
Boulder, CO
grva7970@colorado.edu

***Abstract*—In this project I look at several methods for learning multi-agent policies in the PettingZoo Battle environment. A heuristic policy is simple to implement and produces good results compared to a random policy. This heuristic policy would struggle however against a more intelligent strategy. Q-learning, and Double Q-learning are attempted in this project, however neither were able to adequately learn optimal policies due to the large state and action spaces. Deep Q-learning was implemented in order to train a network to approximate the state-action value function, and achieved promising results. In the future, more advanced methods could be explored to push the creation of more co-operative and effective team strategies.**

I. Introduction and Motivation

Multi-agent reinforcement learning is of interest for many different applications. Games such as tag, tic-tac-toe, and checkers all involve more than one agent interacting in an environment. Learning a strategy for an agent to achieve rewards in such an environment is challenging due to the uncertainty in how other agents will affect the environment.

The PettingZoo API hosts many different multi-agent reinforcement learning environments. In this project the 'Battle' environment will be utilized. This environment has two teams of agents, 'red' and 'blue', which receive rewards from attacking near opponents. In this project, a simple baseline heuristic policy is introduced for the red team, and the performance is tested against that of a random blue policy. Once this heuristic policy is developed, a Deep Q-Network(DQN) is trained to compete against the heuristic policy.

This type of learning can be applied to many different real-world scenarios, where agents need to work collaboratively to achieve the best possible reward, or increase their chances of survival.

II. Background and Related Work

Previous works elaborate on methods to solve some of the challenges introduced in Multi-Agent Reinforcement Learning (MARL). Specifically, work has been released detailing some of the efforts to solve PettingZoo environments, including the Battle environment used in this project. The works that were used to inspire and drive this project are detailed below:

A. Related Works

1) Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms: This work details several different methods for formalizing MARL problems [1]. It outlines 3 main sub-classes for MARL environments – I use the mixed setting class to formulate the Battle environment.

2) Incentivizing Exploration In Reinforcement Learning With Deep Predictive Models: This paper relates to this project due to the common problem of exploring large state and action spaces [2]. While none of the methods specifically mentioned in this paper were deployed in this project, they likely could have improved exploration versus the simple epsilon-greedy method used in this project.

3) PettingZoo: A Standard API for Multi-Agent Reinforcement Learning: This paper introduces the PettingZoo API and the theory behind its implementation [3]. It introduced me to the idea of a "Partially Observable Stochastic Game", which is what they modeled many of their environments as. For simplicity, I stuck with the MDP representation for my problem.

4) MAgent: A Many-Agent Reinforcement Learning Platform for Artificial Collective Intelligence: This work details the implementation of the Battle environment, used in PettingZoo [4]. While not going into many details of their hyperparameter choices, they train the Battle environment very successfully using a variety of methods and determine DQN gave the best results, inspiring the usage of DQN in this project.

5) A Comprehensive Survey of Multiagent Reinforcement Learning: This paper outlines additional strategies for dealing with MARL problems [5]. While non of these specific strategies are used in this project, an interesting extension would be to use the 'Hyper-Q' algorithm they introduce. This algorithm adds the other agent's models as states in the system. The Hyper-Q algorithm then learns over these states, allowing it to better learn a policy partially based on the policy of the other agents. This could potentially introduce better collaboration between agents on the same team if used for this project.

III. Problem Formulation

The Battle environment consists of agents for two-teams, red and blue. Each agent has no knowledge of the intentions of the other agents, other than its color. Further, agents on the same team *do not* receive rewards based on the actions taken from other agents, they only receive rewards from their own actions. Because of this, each agent can be thought of as self-interested, only seeking to take the action which maximizes its expected future reward. This problem can thus be defined as a mixed setting MARL, as we aim to both produce a strategy which competes with the other team, but also helps those on the same team.

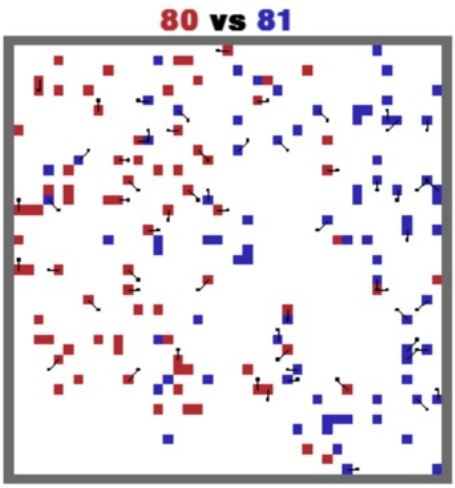

Fig. 1. Rendered frame from the Battle environment

Since each agent makes decisions individually, this environment can be represented as a Markov Decision Process, where each agent lives in its own MDP: $M = \{\mathcal{S}, \mathcal{A}, R, T, \gamma\}$. The state space $\mathcal{S}$ in this MDP relates to the observations seen from an individual agent. The Battle environment has 6 unique input channels, each corresponding to a different element of the environment. To simplify the learning space for this project, only one channel is utilized, the location of nearby enemies. This channel is a 13×13 matrix which represents the cells surrounding the agent. A zero indicates an enemy is not present in that cell, and one indicates there is an enemy present. $\mathcal{S}$ is formally defined as the following vector, where the 13×13 matrix is flattened to 1-dimension:

$$s \in S,\ where\ s = [x_1, x_2..x_{169}],\ x_i \in \{0, 1\}\ for\ i = 1, .., 169$$

There are 21 discrete actions any agent can take, the first 13 actions correspond to movement in the space, and the last 8 actions correspond to attacking different neighboring cells.

$$\mathcal{A} = \{1, 2, ..., 21\}$$

Each action for the agent is deterministic in that the exact action desired is performed, however taking a specific action will not guarantee any particular next state is achieved. This is because the actions taken by the other agents in the environment will influence the next state observed by an agent in addition to its own action. Since this MDP only takes into account the actions of one agent, the environment interaction from the other agents can be treated as transition uncertainty.

The Battle environment has a predefined reward mapping which was not adjusted for this project:

$$R = \begin{cases} -0.005 & \text{if agent takes a step} \\ -0.1 & \text{if the agent dies} \\ -0.1 & \text{if the agent attacks unsuccessfully} \\ +0.1 & \text{if the agent attacks successfully} \end{cases}$$

Each agent begins with 10 HP, and it loses 2 HP every time it gets attacked. Once an agent hits 0 HP it 'dies' and is removed from the environment.

The discount factor γ was chosen to be 0.97 for this project. This was a somewhat arbitrary choice, in-line with a typical discount factor. The discount factor was chosen to be relatively high in order to encourage the agents to strive to attack rather than stay stationary.

IV. Methodology

A. Heuristic Policy

To begin, a heuristic policy was first created. This policy follows three main steps:

- If no agents are in view, take a random movement action
- If one or more enemy agents are in view, but not in attack range, move towards the closest one
- If one or more enemy agents are in attack range, attack the nearest

The resulting heuristic policy is very aggressive in pursuit of defeating the enemy. One potential short-coming of this strategy is that it always move towards the enemy, regardless of whether it may be outnumbered by blue agents. This could lead to it being easily overwhelmed by a more strategic opponent.

B. Q-learning and Double Q-Learning Policies

The first attempt at learning an optimal policy was with Q-learning. The Q-learning algorithm runs for 10,000 episodes, and trains the red team to defeat the blue team. The learning rate, α, for Q is set at 0.1. The red agents act in this environment according to an epsilon-greedy policy. As will be discussed more at-length in the results section, Q-learning did not produce any useful policies for the red team in this project. This is likely due to the extremely large state-action dictionary, which grew to over 60,000 for even a minimized state-space.

Double Q-learning was implemented to attempt to see if better results could be achieved. In double Q-learning, a second Q-table is used in the update step of the learning algorithm. Doing this can improve the stability of the policy learning.

C. Deep Q-Network Learning

I base my DQN implementation on a similar implementation using PyTorch for the OpenAI cartpole environment [6]. The Q-network is constructed to teach the blue team to defeat the red team which takes actions based on the heuristic policy. The constructed policy network has one hidden layer with 64 nodes. The activation function used for this hidden layer is the hyperbolic tan function. The output size of this network is 21, the number of discrete actions an agent can take. To select an action using this policy network, an observation is passed through the network, and the action corresponding to the maximum value returned is the action taken.

The policy is optimized using the ADAM optimizer and the Mean-Squares Error loss function. This optimization is

performed by randomly selecting a (s, a, s', R) tuple from a memory buffer which stores 1000 tuples. The memory buffer is initialized with 128 tuples before optimization of the policy net begins.

The target net that is used to optimize the policy is updated every 10 episodes.

When learning, the agent uses an epsilon-greedy policy to explore the environment. Epsilon begins at 0.99 and decays linearly at a rate of -0.002 per episode. The minimum epsilon value is capped at 0.05.

A summary of the hyper-parameters used to train the Q-network are shown below:

Hyper-parameter	**Value**
Target Update Frequency	10 episodes
Number of Episodes	500 episodes
Epsilon Decrease Rate	-0.002
Minimum Epsilon	0.05
Batch Size	128 steps
Max-steps per Episode	1600 steps

V. Results

The results for the heuristic, Q-learning, and DQN policies are discussed below:

A. Heuristic Policy

The heuristic policy created was tested against a random policy for the opponent. In all of the tests, the heuristic policy was able to defeat all adversaries with 100% survival rate. The mean and standard deviation for the heuristic policy reward accumulated over all red agents is 61.71 ± 30.96. The random policy average reward against the heuristic policy is -9.0 ± 2.18. This means the heuristic policy is able to gain a large advantage over the random opponents, serving as a good baseline strategy to compete against.

B. Q-learning/Double Q-Learning

As mentioned earlier, the Q-learning and Double Q-learning algorithms struggled to learn any sort of policy that improved over the random. Figures 2 shows that no learning trend appears for the regular Q-learning for over 1600 episodes.

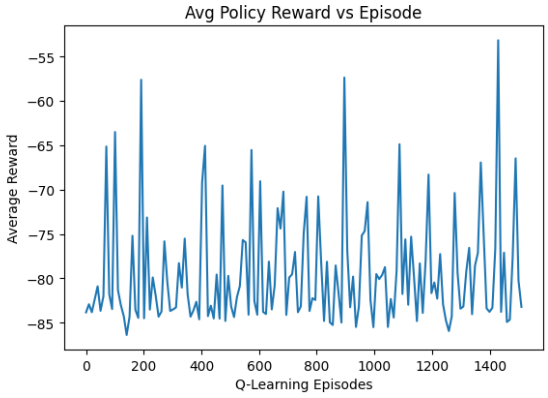

Fig. 2. Q-learning learning curve over 1,600 training episodes

Once it was realized that Q-learning was too limited to learn a good policy for this MARL, Double Q-learning tested, with 10 times the number of training episodes performed(fig 3). This training run took upwards of 6 hours to complete however it still appeared that the learning algorithm was too limited to learn over the large state-space of the Battle environment.

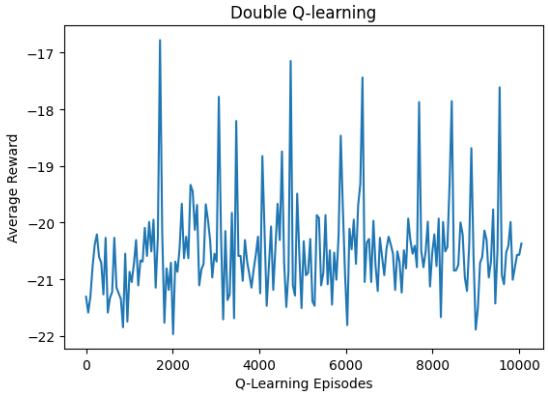

Fig. 3. Double Q-learning learning curve over 10,000 episodes

C. DQN

The DQN implementation was trained over 500 episodes, and took roughly 3.5 hours to complete. The learning curve for this training is displayed in figure 4.

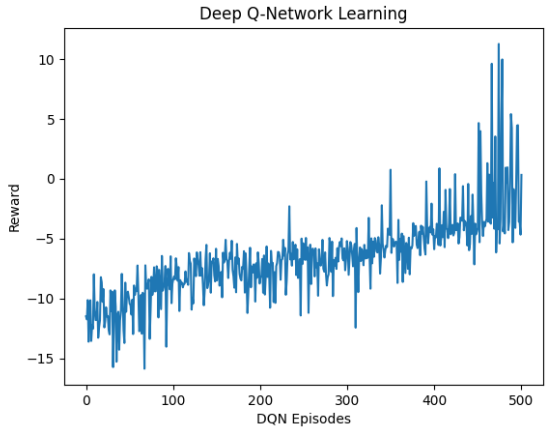

Fig. 4. Policy reward versus episode number for DQN

Here there is a clear upwards trend in average reward, with rewards reaching over 0 and upwards of +10 by the end of the training. This shows DQN was very successful in leaning a policy to compete against the heuristic policy. The survival duration of the DQN agents are plotted in figure 5.

Again, there is a clear trend in this figure indicating that the DQN agents were learning an effective strategy to survive against their opponents longer. Due to time constraints, no more than 500 episodes were used to train the DQN. These results suggest however that with more training episodes, the DQN agents would become more and more successful.

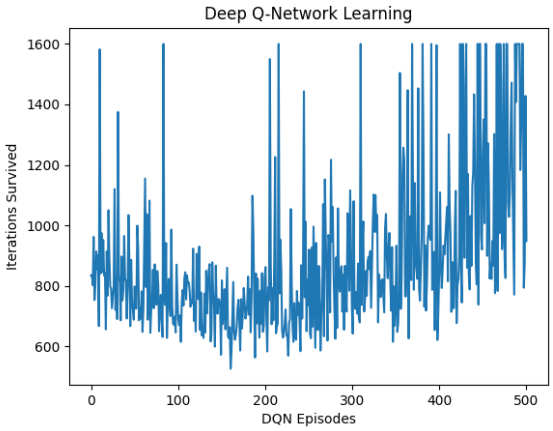

Fig. 5. Number of iterations survived against heuristic policy versus episode number

The average reward is -1.308 ± 4.7, and the average survival length is 1245.3 ± 285.7.

D. Discussion/Comparison

The results of the different policies are summarized in the table below(Q-learning not included):

Policy	Opponent	Reward	Survived Iterations
Random	Heuristic	-9.0±2.2	866±147.8
DQN	Heuristic	-1.308 ±4.7	1245±284.7
Heuristic	Random	61.71±30.9	1600±0

When observing the learned DQN policy compete against the heuristic, it executes an interesting strategy to maximize survival rate. This is visualized in the sequential frames below(figs 6, 7):

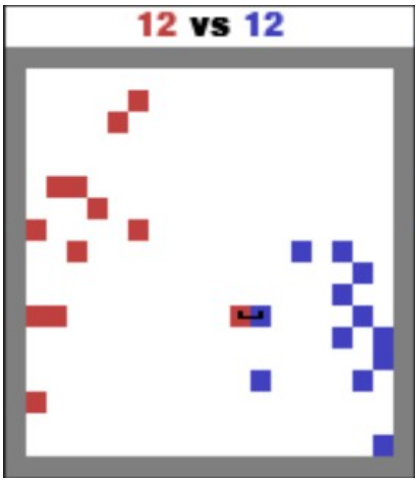

Fig. 6. Early stages of DQN(blue) vs Heuristic

In figure 7, the DQN agents produce a formation in the lower right corner of the environment. This is far enough away from the red agent's observation zone, so they exist safely there until the red agents take a random movement which

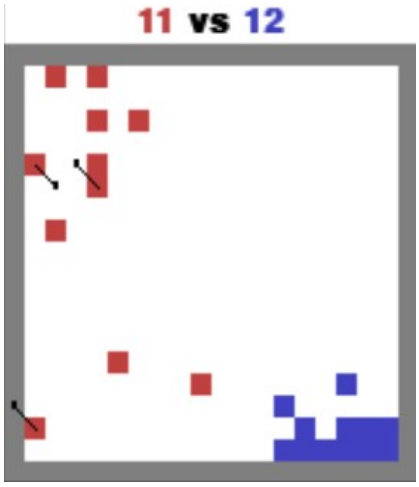

Fig. 7. Later stages of DQN vs heuristic

makes the blue agents discoverable. The occasional positive reward returned by the DQN policy indicates that the policy has learned how to attack successfully on occasion, however from reviewing their interactions, these attack are performed sparingly and are not always accurate. Without an effective attack strategy, the DQN policy still succumbs to the heuristic policy after enough time.

VI. Conclusion

In this project, several methods of reinforcement learning for the multi-agent Battle environment were attempted, to varying levels of success. The simple Q-learning and Double Q-learning algorithms severely struggled at learning the state-action values due to the very large number of unique state-action pairs.

Deep Q-learning proved to be an effective strategy at learning in this large state-action space, and actually produced a policy which could compete with the heuristic policy.

There are many additional aspects that I would have liked to have implemented in my project had there been more time. More profound results would have required benchmarking for a set of varying hyper-parameters but due to the very long training times, this was infeasible for my setup. I would have liked to have explored the impact that adding additional observation channels to the state would have had, such as adding the location of teammates to the state-space. Doing so could have introduced teamwork strategies into the DQN policy.

VII. Release/Contributions

This project was completed individually and I give permission to post this report publicly. My code for this project can be found at this Github address: https://github.com/griffjv/multi_agent_ml.

References

[1] Zhang, Kaiqing, Zhuoran Yang, and Tamer Başar. "Multi-agent reinforcement learning: A selective overview of theories and algorithms." Handbook of Reinforcement Learning and Control (2021): 321-384.

[2] Stadie, Bradly C., Sergey Levine, and Pieter Abbeel. "Incentivizing exploration in reinforcement learning with deep predictive models." arXiv preprint arXiv:1507.00814 (2015).
[3] Terry, J., et al. "Pettingzoo: Gym for multi-agent reinforcement learning." Advances in Neural Information Processing Systems 34 (2021): 15032-15043.
[4] Zheng, Lianmin, et al. "Magent: A many-agent reinforcement learning platform for artificial collective intelligence." Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 32. No. 1. 2018.
[5] L. Busoniu, R. Babuska and B. De Schutter, "A Comprehensive Survey of Multiagent Reinforcement Learning," in IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), vol. 38, no. 2, pp. 156-172, March 2008, doi: 10.1109/TSMCC.2007.913919.
[6] "Reinforcement Learning (DQN) Tutorial¶." Reinforcement Learning (DQN) Tutorial - PyTorch Tutorials 1.11.0+cu102 Documentation, https://pytorch.org/tutorials/intermediate/reinforcementqlearning.html.