# AIT3007 Final Project Report: Group 6

1st Vương Ngọc Quân
*22022616*

2nd Hoàng Việt Tùng
*22022663*

3rd Phạm văn Trường
*22022564*

## I. Introduction

In this report, we discuss about the application of Deep Q-Networks (DQN) in MAgent2 framework. Our implementation integrates advanced techniques such as Double Q-Learning and Prioritized Experience Replay. This report presents our solution for the final project of the class *2425I_AIT3007_37*.

Summarize results: To evaluate our approach, we were provided with three pre-trained policies of different levels by our professor. We aim to beat these pre-trained teams of agent with our proposed approach. The evaluation result shows that our implementation of DQN is extremely effective as we successfully beat all three pre-trained policies with a win rate of 100%.

Bảng I
Main results. You can report with std for better evaluation

| Model | Win | Draw | Lose |
|---|---|---|---|
| Random | $100 \pm 0$ | $0 \pm 0$ | $0 \pm 0$ |
| Pretrain-0 | $100 \pm 0$ | $0 \pm 0$ | $0 \pm 0$ |
| New Pretrain | $100 \pm 0$ | $0 \pm 0$ | $0 \pm 0$ |

Average score for each models, using 1 point for Win, 0.5 point for Draw and 0 point for Lose:

- Random: 30/30.
- Pretrain-0: 30/30.
- New Pretrain: 30/30.

## II. Methods

### A. Q-Learning for Multi-Agent Reinforcement Learning Systems

Q-learning, a value-based Reinforcement learning algorithm, where each agent maintains a Q-function that estimates the expected cumulative reward for taking an action in a given state. Therefore constructed a optimal policy by maximizing the expected cumulative reward.The Q-function is updated iteratively using the Bellman equation as follows:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left( r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right)$$

where $s$ and $s'$ are the current and next states, $a$ and $a'$ are actions, $r$ is the immediate reward, $a$ is the learning rate, and $\gamma$ is the discount factor.

In this experiment, our approach is implemented in MAgent2 battle_v4 environment, where there will be 2 teams of *red* and *blue* agents in a large-scale team battle. Each agent will be rewarded for their individual performance and not for the performance of their neighbors. Each agent has 10 HP, are damaged 2 HP by each attack, and slowly regain 0.1 HP every turn, so it is best to kill an opposing agent quickly.

### B. Deep Q-Network (DQN) in MAgent2 Environment

To handle states and action spaces, Q-functions are approximated using neural networks. Our approach employed Deep Q-Networks (DQNs), where:

- The input is the state representation, here, each agent at each time step will have a observation of shape *(height, width, channels)* which equivalent to the state representation. In default setting, the observation shape is *(13, 13, 5)* for each agent.
- The output is a vector of Q-value for all possible actions. The network is trained to minimize the temporal difference (TD) error:

  $$L(\theta) = \mathbb{E}_{(s,a,r,s')} \left[ \left( r + \gamma \max_{a'} Q_{\text{target}}(s', a'; \theta') - Q(s, a; \theta) \right)^2 \right]$$

  where $\theta$ and $\theta'$ are the parameters of the primary and target Q-networks, respectively.

### C. Prioritized Experience Replay (PER)

Prioritized Experience Replay (PER) is used in our approach with a mechanism for prioritizing experiences based on their Temporal Difference (TD) errors. The core idea is that not all experiences with higher TD error allows the agent to learn more effectively from the mistakes that have the largest impact on its performance, so giving experiences their corresponding prioritized weights will able us to filter out unnecessary experiences.

In our approach, this prioritization is controlled by a hyperparameter, as we called it *buffer_alpha*, which determines the level of emphasis placed on high-priority experiences. We also leverage importance sampling, which aims to correct for any bias introduced by the prioritization process, ensuring that the updates during training remain fair and unbiased.

By using PER with importance sampling, we expect our team of agents learn faster and more efficiently as we assumed that certain experiences have a much larger influence on the agent's behavior than others in an environment like MAgent2's *battle_v4*.

### D. Double Q-Learning

The traditional Q-Learning methods often suffer from overestimation bias, where the action-value function tends to overestimate the value of certain action. This is caused by the use of the same values to both select and evaluate actions,

leading to unrealistic overestimates that can negatively impact learning stability and overall performance.

To address this issue, Double Q-learning proposes a two-network approach, where two separate Q-value estimations are maintained. Specifically, one Q-function is used to select the action, while the other is used to evaluate the value of that action, we called it the target network. This separation reduces the overestimation bias by decoupling action selection from evaluation.

During this experiment, we observed that by applying Double Q-learning, the learning process is more stable and accurate. This leads to better performance, especially in MAgent2's *battle_v4* environment, where we have to handle multiple action spaces from multiple agents.

### E. Epsilon-Greedy Action Selection

An agent must explore its environment sufficiently to discover optimal actions while exploiting its current knowledge to maximize rewards. This makes epsilon-greedy strategy a fundamental approach to balancing exploration and exploitation in Reinforcement Learning.

As usual, our approach used a $\epsilon$ variable as the exploration rate, which means at each time-step, an agent will have a probability of $\epsilon$ to take an random action. This value is decayed over time, as we encourage our team of agents to explore in the early stages of learning and gradually decreasing to prioritize exploitation as the team becomes more confident in their value estimates.

## III. Implementation

### A. Environment Setup

As provided pre-trained policies will be evaluated with our approach in tuned setting of MAgent2's *battle_v4*, combined with our observation during the time of experiment, we find that tuning these environment's parameters significantly improve our team's performance. With default settings of the environment, our agents tend to "hide" as them are penalized heavily for trying to attack anything but very lightly for "doing nothing" or "running". The penalty for being dead is also low compared to the reward for successfully attempts to attack an opponent. With that being said, our tuned training script set a positive value for the step reward, penalize our agents more for dying and give them more reward for successfully attacking an opponent. Our result shows that our trained team proactively find for their opponent, attack them, while still trying to stay alive.

In contrast, although we will "fight" pre-trained teams in 300 time-steps setup, we still set this parameter to the default value, which is 1000, as we assumed this will help our agents explore more scenarios during the training process.

Our settings for the environment is described as follow:

- **map_size**: 45 - Dimension of the (square) map.
- **step_reward**: -0.005 (default) and 0.01 (tuned) - Reward after every step.
- **dead_penalty**: -0.1 (default) and -0.8 (tuned) - Reward when killed.
- **attack_penalty**: -0.1 - Reward when attacking anything.
- **attack_opponent_reward**: 0.2 (default) and 0.5 (tuned) - reward added for attacking an opponent.
- **max_cycles**: 1000 - Number of frames (a step for each agent) until game terminates.

### B. Q-Network Architecture

We designed our Q-Network to process observation of each agent and output it's action-value estimate. We leverage convolutional layers and fully connected layers to effectively extract spatial and high-level features from input observations. Our Q-Network architecture including these components:

*1) Convolutional Layers:* Two convolutional layers are used to extract features from the input observation. The feature map depth will be increased from 32 channels in the first layer to 64 channels in the second,

*2) Fully Connected Layers:* The output of convolutional layers then is flattened and go through two fully connected layers to map the extracted features to action-value predictions. The first fully connected layer reduces the dimensionality to 128 neurons, providing a dense representation of the input features. The second layer outputs action values corresponding to the available action, which results in a list of value for each possible action for a given observation.

*3) Activation Functions:* We applied Rectified Linear Unit (ReLU) activation functions after each convolutional and fully connected layer. ReLU introduces non-linearty, allowing our network to learn more complex pattern in the input observation.

### C. Prioritized Replay Buffer

Our implementation of PER leverages a SumTree data structure as the backbone for our custom Prioritized Replay Buffer which helps to manage and sample experiences based on their priority.

During our experiment, we used both normal Replay Buffer, which simply save and sample experiences with no prioritization, and Prioritized Replay Buffer, we observed that using Prioritized Replay Buffer absolutely outperform normal Replay Buffer in the same setting.

Our implementation of SumTree:

- **Tree Structure**: A binary tree structure where leaf nodes represent experience priorities, and internal nodes store cumulative priorities.
- **Priority Update**: When a priority is updated, changes propagate up the tree to maintain accurate cumulative priorities.
- **Sampling**: This is done by mapping a random value to a specific leaf node, enabling proportional sampling based on its priority value.

Our implementation of PrioritizedReplayBuffer:

- **Prioritization Parameter *(buffer_alpha)***: Degree of prioritization.

- **Storage**: Store experiences (state, action, reward, next state, done flag) along with their priorities in the SumTree.
- **Weighted Sampling**: Priorities are converted in to sampling probabilities during sampling, which makes higher priorities are sampled more often.
- **Importance Sampling Weights**: This helps ensure that updates is not biased by non-uniform sampling.

Our PrioritizedReplayBuffer has the same use as the ReplayBuffer with functionalities like adding experiences, batch sampling with the ability to update experiences' priorities.

### D. Training Process

For each team of *red* and *blue* agent, we will have two networks as mentioned. Each team takes actions determined by epsilon-greedy policy from their respective networks.
Our implemented PrioritizedReplayBuffer store experiences for each team at each time-steps after both team have taken their actions.
The learning process happens with batch updates once the buffer size meets the threshold, with these steps:

- Compute target Q-values using Double Q-learning as mentioned above.
- Update the TD errors in the buffer.
- Minimize weighted TD loss and backpropagate using team-specific optimizers.

Our hyperparameters like *epsilon* used for selecting actions and *beta* used for sampling is also updated after each training episode

## IV. Experiment Results

### A. Hyperparameters Selection

Prioritized Replay Parameters:

- **buffer_alpha**: 0.6 - Ensuring experiences with higher TD errors are sampled more frequently while still keeping balance between uniform and prioritized sampling.
- **beta**: 0.4 - Starts low to reduce impact of importance sampling early in training as it is not very accurate early on.
- **beta_increment**: 1e-3 - Gradually increases *beta* to 1.0 over training, ensuring convergence.

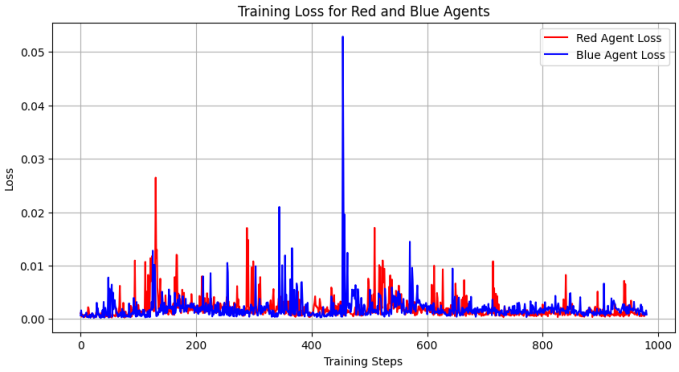
**learning_rate**: 0.0005 - Ensure stable convergence. **buffer_size**: 10000 - large enough to store diverse experiences but manageable in terms of memory usage.
Exploration Parameters:

- **epsilon**: 1.0 - Full exploration at first to gather experiences.
- **epsilon_decay**: 0.998 - Decay rates to ensure exploitation later on.
- **epsilon_min**: 0.5- Maintains some level of exploration even later on.

**gamma**: 0.98 - Discount factor, set slightly below 1.0 to emphasize short-and medium-term rewards while long-term rewards still matter.
**tau**: 0.005 - Soft update coefficient, slow updates to the target network.

### B. Loss result

Since we trained both *red* and *blue* teams, with the settings and steps mentioned above, we have successfully trained two "quite" decent teams of agents, which we only saved *blue* model to evaluate.
However, here is the loss result of our training process after 50 episodes, which is executed in just 45 minutes of training on Kaggle:




## V. Conclusion

In this experiment, we have successfully implemented DQN with advanced techniques such as Double Q-Network and PER. By combining these techniques, we were able to improve our model's performance, as shown by our trained agent defeating all three provided pre-trained policies in the environment.
This result demonstrates how these methods can help solve complex problems in dynamic and competitive situations like MAgent2's battle_v4 environment.
During experiment, we have also tried to implement various Multi Agent Reinforcement Learning algorithms such as Dueling Q-Network, Deep-Graph Network, Multi Agent Deep Deterministic Policy Gradient,... However, due to the time limit of this project, and the complexity of those algorithms, we are still not able to finish implementing them.
Therefore, after this project, we aim to continue implementing new algorithms for this problem, experimenting various methods to find out which method is the best and truly "undefeated" in this environment. With that being said, our implementation of Deep Q-Network and other algorithms in the future will be on this repo https://github.com/HvTung04/RL-final-project-AIT-3007.

## Tài liệu


[1] G. Eason, B. Noble, and I. N. Sneddon, "On certain integrals of Lipschitz-Hankel type involving products of Bessel functions," Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955.
[2] Roderick M, MacGlashan J, Tellex S. Implementing the deep q-network. arXiv preprint arXiv:1711.07478. 2017 Nov 20.
[3] Van Hasselt, Hado, Arthur Guez, and David Silver. "Deep reinforcement learning with double q-learning." Proceedings of the AAAI conference on artificial intelligence. Vol. 30. No. 1. 2016.
[4] Schaul, Tom. "Prioritized Experience Replay." arXiv preprint arXiv:1511.05952 (2015).
[5] Chadi, Mohamed-Amine, and Hajar Mousannif. "Understanding reinforcement learning algorithms: The progress from basic Q-learning to proximal policy optimization." arXiv preprint arXiv:2304.00026 (2023).