Reinforcement Learning for Multi-Agent Environments: A Survey on MAgent2 Battle Using DQN Variants Report for AI3007: Group 22

Nguyen Huy Hoang, Tran An Thang, Pham Dang Phong
University of Engineering and Technology,
Vietnam National University,
Hanoi, Vietnam
{22022584, 22022525, 22022614}@vnu.edu.vn

***Abstract*—This report will discuss reinforcement learning techniques applied to multi-agent environments (MAgent2 Battle). We present three main models that will be discussed: Deep Q-Network (DQN) with epsilon-greedy exploration policy, DQN with Noisy Networks, and Value Decomposition Networks (VDN) with hypernetwork mixers. All the above models are trained and evaluated against three different adversaries: random agents, first pre-trained agents, and last pre-trained agents. After analyzing the performance differences between them, we find that the Noisy Network-based DQN model has a faster learning speed and more stable performance than the greedy epsilon one. In addition, the VDN mixer is also effective in coordinating the actions of the agents and achieves impressive performance by defeating three different opponents with a win rate of 1.0. Going deeper into the model, we improve the learning process of the agents compared to the traditional Q-network for the multi-agent problem by incorporating advanced techniques such as Dueling Q-Network and Prioritized Experience Replay. This paper provides an overview of the application of these methods in multi-agent environments in general and its effectiveness in reinforcement learning in particular.**

***Index Terms*—Reinforcement Learning, Multi-Agent Systems, Deep Q-Network (DQN), Epsilon-Greedy, Noisy Networks, Value Decomposition Networks (VDN), Dueling Q-Network, Prioritized Experience Replay, MAgent2 Battle**

I. INTRODUCTION

Reinforcement Learning (RL) is a branch of Machine Learning (ML), where agents can learn to respond to the environment to get a goal based on the data they receive from the environment. And in multi-agent environment, Multi-Agent Reinforcement Learning (MARL) is a suitable method due to its stability and interaction between agents.

In MARL, Independent Q-Learning (IQL) is a popular method, in which each agent learns its own Q-function. IQL is easy to implement,but it is unstable and has poor convergence, because the environment continuously changes when other agents learn. So we have more advanced ways, such as Value-Decomposition Networks (VDN), can fix these limitations by decomposing the group value function into individual agent value functions to improve coordination and credit allocation.

In this report, we will find out about the application of some RL algorithms in the MAgent2 Battle environment. For more information, the Battle scenario in MAgent2 simulates two teams of agents fighting each other. Each agent has only limited observation and can take actions such as moving (go left, go right, keep direction) and attacking. Agents can get reward based on their actions. Agents slowly regain HP over time, so it is best to kill an opposing agent quickly. Each agent has 10 HP, loses 2 HP each time it is attacked, but recovers 0.1 HP after each turn. And agents can only perform one action per turn, either move or attack. Attacking an ally will not count.

To develop the training program, we focus on three main models: a standard Deep Q-Network (DQN) with an epsilon-greedy exploration policy ; a better version of DQN using Noisy Networks to enhance exploration ; and a Value Decomposition Network (VDN) using a Hypernetwork Mixer to optimize coordinated action selection. In addition, we also apply some advanced techniques such as Dueling Q-Network and Prioritized Experience Replay to improve learning efficiency.

The goal of this study is to compare the performance and the stability of these models when trained against opponents of different skill levels:

- **Random agents - Lv1**: Opponents that choose actions randomly.
- **Pretrained agents - Lv2**: Opponents with basic strategic capabilities.
- **Final pretrained agent - Lv3**: A more sophisticated pretrained agent, representing a higher level of challenge.

Our results show that the applying of Noisy Networks into the DQN leads to more efficient learning while training model. This method achieves a good balance between exploration and exploitation without any exploration policy like epsilon-greedy. Furthermore, we verify that using the VDN model with a hypernetwork mixer leads to improve performance by enabling effective coordination among agents achieves the highest results among trained models.

Summarize results:

TABLE I
MAIN RESULTS

Method	Opponent Type	Win Rate (Red)	Win Rate (Blue)	Draw	Average Reward	
					Red	Blue
Dueling DQN Epsilon	Random Policy	0.0	1.0	0.0	-2.2322	4.4912
	Trained Policy	0.0	1.0	0.0	0.9860	4.8346
	Final Trained Policy	0.8333±0.1	0.1333±0.1	0.0334	4.2722	3.5759
Dueling DQN Noise	Random Policy	0.0	1.0	0.0	-0.8878	4.9122
	Trained Policy	0.0	1.0	0.0	1.7586	4.9438
	Final Trained Policy	0.2333±0.1	0.4±0.1	0.3667	4.3527	4.5171
VDN Mixing	Random Policy	0.0	1.0	0.0	-0.8729	4.8768
	Trained Policy	0.0	1.0	0.0	2.5829	4.9884
	Final Trained Policy	0.0	1.0	0.0	2.8536	4.8611

II. METHODOLOGY

This study focus three core models: the standard Deep Q-NetWork with epsilon-greedy policy, a Deep Q-Network with Noisy Networks, and a Value Decomposition Network with a hypernetwork mixer. To increase performance, those models are integrated with techniques such as the Dueling Q-Network architecture and Prioritized Experience Replay.

A. Models

1) Deep Q-Network (DQN) with Epsilon-Greedy: This model will approximate the optimal action-value function $Q^*(s, a)$ by using a deep Q network. The Q-function estimates the expected cumulative reward of taking action a in state s and following a optimal policy. The network is trained to minimize the temporal difference (TD) error, which is the difference between the predict Q-value and the target Q-value.

The target Q-value is calculated by using the Bellman equation:

$$Q(s, a) = r + \gamma \max_{a'} Q(s', a'; \theta^-) \tag{1}$$

where:

- s: the current state.
- a: the action in the current state.
- r: the immediate reward after taking action a in state s.
- s': the next state - after state s
- a': the action that maximizes the Q-value in the next state.
- γ: the discount factor.
- θ^-: the parameters of the target network.

a) Epsilon-Greedy Exploration.: To balance between exploration and exploitation, the epsilon-greedy policy is used. With probability ϵ, a random action is choosen in each agent; otherwise, agent will choose action that has the highest Q-value:

$$a = \begin{cases} \text{random action} & \text{if } random < \epsilon \\ \arg\max_a Q(s, a; \theta) & \text{otherwise} \end{cases} \tag{2}$$

In the start, ϵ is settled high value (e.g., 1.0) and gradually decays to a minimum value (e.g., 0.1).

2) DQN with Noisy Networks: Noisy Networks improve exploration by adding noise parameters directly into the network's weights and biases

A Noisy Linear layer is defined as:

$$y = (\mu^w + \sigma^w \odot \epsilon^w)x + (\mu^b + \sigma^b \odot \epsilon^b) \tag{3}$$

where:

- x: the input.
- y: the output.
- μ^w and σ^w: the learnable mean and standard deviation parameters for the weights.
- μ^b and σ^b: the learnable mean and standard deviation parameters for the bias.
- ϵ^w and ϵ^b: random noise.

a) Advantages of Noisy Networks.:

- **State-dependent exploration:** The noise is added to the weights depends on the input state, allowing for more exploration.
- **Learnable exploration:** The network will learn the optimal level of exploration during training.
- **Improve stability and speed:** Noisy Networks often lead to faster and more stable learning than epsilon-greedy policy.

3) Value Decomposition Networks (VDN) with Hypernetwork Mixer: VDN solves the problem of multi-agent coordination by decomposing the team value function into individual agent value functions. The total Q-value is approximated as the sum of individual agent Q-values:

$$Q_{\text{total}}(s, \mathbf{a}) = \sum_i Q_i(s_i, a_i) \tag{4}$$

where:

- Q_{total}: the total Q-value for the team (blue team / red team).
- s: the global state.
- $\mathbf{a} = (a_1, a_2, \ldots, a_n)$: the action.
- s_i: the local observation of agent i.
- a_i: the action taken by agent i.
- Q_i: the individual Q-function for agent i.

a) Hypernetwork Mixer.: We improve the VDN model by using a hypernetwork mixer, which creates the weights of a mixing network to combine individual Q-values based on the global state:

$$Q_{\text{total}}(s, \mathbf{a}) = f_{\text{mixer}}(Q_1(s_1, a_1), Q_2(s_2, a_2), \ldots, Q_n(s_n, a_n), s; \theta_{\text{mixer}}) \tag{5}$$

where:

- f_{mixer}: the mixer network.
- s: the global stat.
- θ_{mixer}: the parameters of the hypernetwork.

b) Advantages of Hypernetwork Mixer.:

- **Context-aware coordination:** Agent can learn the complex, non-linear relationships between individual Q-values and the global state.
- **Flexibility**

B. Advanced Techniques

*1) **Dueling Q-Network**:* The Dueling Q-Network improves the Q-function approximation by dividing it into two streams:

1) Value Stream: $V(s)$, which estimates the state value.
2) Advantage Stream: $A(s, a)$, which estimates the advantage of action a in that state s.

The Q-value is then calculated as:

$$Q(s, a) = V(s) + \left(A(s, a) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s, a') \right) \quad (6)$$

where $|\mathcal{A}|$ is the number of actions.

*2) **Prioritized Experience Replay**:* Prioritized Experience Replay (PER) improve learning efficiency by sampling experiences from the replay buffer based on their priority. The priority of an experience is determined by the size of the temporal difference (TD) error:

$$\text{Priority} = |\text{TD Error}| + \epsilon \quad (7)$$

where ϵ is a small positive constant to ensure sampling probability is non-zero.

a) Sampling Probability.: The probability of sampling experience i is:

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha} \quad (8)$$

where p_i is the priority and α controls prioritization.

b) Importance Sampling.: To correct bias, importance sampling weights are used:

$$w_i = \left(\frac{1}{N} \cdot \frac{1}{P(i)} \right)^\beta \quad (9)$$

where N is the size of replay buffer and β anneals from β_0 to 1 during training.

C. Environment

The MAgent2 Battle environment has two teams of agents (red and blue) fighting head-to-head. Each agent has a limited field of view (adjustable to unlimited) and can only perform one of the actions such as moving and attacking each turn. This is a limited-observability, highly random environment with a lot of state and action , which creates challenges for developing and testing algorithms.

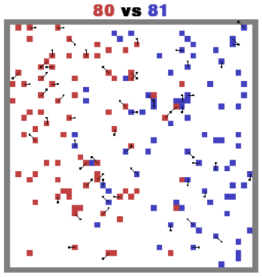

Fig. 1. MAgent2 battle

The observation space is a 13x13 map with the below channels (in order):

TABLE II
AGENT OBSERVATION SPACE

Feature	Number of channels
Obstacle/off the map	1
My team presence	1
My team HP	1
My team minimap	1
Other team presence	1
Other team HP	1
Other team minimap	1
Binary agent ID	10
One-hot action	21
Last reward	1
Agent position	2

The global state space is a 45x45 map. It contains the following channels, which are (in order):

TABLE III
GLOBAL STATE SPACE

Feature	Number of channels
Obstacle map	1
Team 0 presence	1
Team 0 HP	1
Team 1 presence	1
Team 1 HP	1
Binary agent ID	10
One-hot action	21
Last reward	1

III. IMPLEMENTATION

The implementation divided into the following key components: agent architecture, learning algorithms, and training procedures.

A. Agent Architecture

Three primary agent architectures are implemented:

1) Dueling DQN with Epsilon-Greedy::

- The observation space, a 13x13x20 map, is processed through a convolutional neural network (CNN) comprising two convolutional layers, each followed by a ReLU activation function.
- The output of the CNN is flattened and fed into two separate fully connected streams:
  - Value stream: A fully connected layer with 120 units followed by a ReLU activation, and a final linear layer with a single output representing the state value, $V(s)$.
  - Advantage stream: A fully connected layer with 120 units followed by a ReLU activation, and a final linear layer with $|A|$ outputs, representing the advantage of each action, $A(s, a)$.
- The Q-value for each action is then calculated as:

$$Q(s, a) = V(s) + \left(A(s, a) - \frac{1}{|A|} \sum_{a'} A(s, a') \right)$$

where $|A|$ is the number of possible actions.

- Epsilon-greedy exploration is employed, where the agent selects a random action with probability ϵ and the action with the highest Q-value with probability $1-\epsilon$. ϵ linearly decays from 1.0 to 0.1 over the course of training.

2) Dueling DQN with Noisy Networks::

- This architecture is similar to Dueling DQN with Epsilon-Greedy, except that the fully connected layers in both the value stream and advantage stream are replaced with NoisyLinear layers, thereby the model prioritizes more exploration during the learning process, achieving higher and more stable performance than Epsilon-Greedy
- NoisyLinear layers incorporate learnable noise parameters (μ and σ) into the weights and biases, enabling state-dependent exploration without the need for an external exploration policy like epsilon-greedy. The output of a NoisyLinear layer is given by:

$$y = (\mu_w + \sigma_w \odot \epsilon_w)x + (\mu_b + \sigma_b \odot \epsilon_b)$$

where x is the input, y is the output, μ_w and σ_w are the learnable mean and standard deviation parameters for the weights, μ_b and σ_b are the learnable mean and standard deviation parameters for the bias, and ϵ_w and ϵ_b are random noise.
- The noise parameters are initialized with a standard deviation (σ_{init}) of 0.5.

3) VDN Mixer with Hypernetworks::

- Each agent employs the same Dueling Noisy Q-Network architecture as described above to compute individual Q-values, $Q_i(s_i, a_i)$, based on its local observation s_i.
- A hypernetwork is used to generate the weights of a mixing network. The hypernetwork takes the global state s (mean of agent's positions) as input and outputs the weights and biases of the mixing network. This can be represented as:

$$W_1 = |\text{hyper\_w\_1}(s)|$$

$$b_1 = \text{mean}(Q_i(s_i, a_i))$$

$$W_{\text{final}} = |\text{hyper\_w\_final}(s)|$$

- The mixing network then combines the individual Q-values into a total Q-value, $Q_{\text{total}}(s, a)$, based on the weights generated by the hypernetwork. The mixing function is implemented using a two layer feedforward network, where the hidden layer activation is RELU and the output layer has no activation.

$$h = \text{RELU}(b_1 + W_1 * Q_i(s_i, a_i))$$

$$Q_{\text{total}} = h * W_{\text{final}}$$

B. Learning Algorithms

The following learning algorithms are utilized:

1) DQN:

- The agents are trained using the Deep Q-Network (DQN) algorithm.
- The loss function is the Smooth L1 Loss between the target Q-value and the current Q-value:

$$L(\theta) = \mathbb{E}\left[\text{SmoothL1Loss}\big(y_i, Q(s, a; \theta)\big)\right]$$

where $y_i = r + \gamma \max_{a'} Q(s', a'; \theta^-)$ is the target, r is the immediate reward, γ is the discount factor, θ are the parameters of the Q-network, and θ^- are the parameters of the target network.
SmoothL1Loss is defined as:

$$\text{SmoothL1Loss}(x, y) = \begin{cases} 0.5(x-y)^2 & \text{if } |x-y| < 1 \\ |x-y| - 0.5 & \text{otherwise} \end{cases}$$

- The target network is updated using Polyak averaging with a tau value of 0.005.

2) Prioritized Experience Replay (PER):

- PER is implemented to improve the efficiency of learning by sampling experiences from the replay buffer based on their priority. The priority of an experience is determined by the absolute temporal difference (TD) error:

$$\text{Priority}_i = |r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta)| + \epsilon$$

where ϵ is a small constant to ensure non-zero probabilities.
- The sampling probability of an experience is proportional to its priority raised to the power of α (set to 0.6):

$$P(i) = \frac{\text{Priority}_i^\alpha}{\sum_k \text{Priority}_k^\alpha}$$

- Importance sampling weights, w_i, are used to correct the bias introduced by non-uniform sampling:

$$w_i = \left(\frac{1}{N} \cdot \frac{1}{P(i)}\right)^\beta$$

where N is the replay buffer size and β is a hyperparameter that anneals from $\beta_{\text{start}} = 0.4$ to 1.0 over the course of training.

C. Training Procedure

1) **Initialization:**
   - Initialize the Q-network, target network, and VDN mixer (if applicable) with random weights.
   - Initialize the replay buffer with a capacity of 500,000.
2) **Training Loop:**
   a) For each episode:
      i) Reset the environment and obtain the initial observations for all agents.
      ii) For each step:
         A) Select actions for each agent based on the chosen exploration strategy (epsilon-greedy or Noisy Networks).

B) Take the selected actions in the environment and observe the rewards, next observations, and termination signals.
C) Store the experience $(s, a, r, s', \text{done})$ in the replay buffer.
D) If the replay buffer contains at least a batch size (4096) of experiences, sample a batch of experiences.
E) Calculate the target Q-values using the target network.
F) Calculate the TD errors and update the priorities in the replay buffer.
G) Calculate the loss using the appropriate loss function (DQN or VDN).
H) Perform backpropagation and update the Q-network parameters using the Adam optimizer with a learning rate of 0.0005.
I) Update the target network using Polyak averaging.

iii) Log the total episode reward and other relevant metrics to TensorBoard.

3) **Evaluation:**

- Every 5 episodes, evaluate the current policy against:
  a) A random agent.
  b) A pre-trained agent (level 2 :red.pt).
- Save the model checkpoint if the average validation reward exceeds the best validation reward seen so far.

D. Hyperparameters

Hyperparameter	Value
Learning rate	0.0005
Discount factor (γ)	0.99
Number of episodes	250(DQN),500(VDN)
Maximum steps per episode	1000
Checkpoint interval	5
Polyak tau	0.005
Replay buffer size	500000
Batch size	512(DQN+ϵ), 4096(VDN)
PER alpha	0.6
PER beta start	0.4
PER beta frames	100000
Noise std_init	0.5
Number of validation episodes	5
Epsilon start	1.0
Epsilon end	0.1

Note: Since the buffer can store even suboptimal values, for dqn, choosing a small batch size will help the model learn faster and converge faster from the first step. For models using noisy networks, since the noise is added so the values are almost the same in the buffer, adding a large batch size will only train faster.

E. Implementation Notes

- The implementation uses PyTorch for building and training the neural networks on Kaggle(GPU P100).
- The MAgent2 library provides the Battle environment and handles agent-environment interactions.
- TensorBoard is used for logging training progress and visualizing performance metrics.
- A separate pre-trained model is loaded for the red agents in the "pretrained policy" evaluation setting.

IV. EXPERIMENTAL RESULTS

A. Opponents

The agents were trained and evaluated against three types of opponents:

1) **Random Policy:** Agents that choose actions randomly.
2) **Pretrained Policy:** Agents with basic strategic capabilities.
3) **Final Trained Policy:** A more sophisticated pretrained agent representing a higher level of challenge.

B. Parameter selection and model training process

We trained three models in the Kaggle environment; the average training time for the models until training completion is given by the table below:

TABLE IV
AVERAGE TRAINING TIME OF MODELS

Model	Training Time (minutes)
Dueling Q Network	45
Dueling Q Network (Noise Network)	45
VDN Mixer	85

C. Evaluation Metrics

Performance was evaluated based on the following metrics:

- **Win Rate:** The proportion of episodes where the blue team wins.
- **Average Rewards:** The average cumulative reward per agent per episode for both red and blue teams.

D. Performance Comparison

Table V summarizes the performance of the three trained blue agents (Dueling DQN with Epsilon-Greedy, Dueling DQN with Noisy Networks, and VDN Mixing) against different opponent types: random policy, pretrained policy, and final trained policy.

Key Observations from Table V:

All three models trained above easily beat random and basic opponents, showing that the models are quite strong against neutral opponents. Notably, both Noisy DQN and VDN Mixing models perform much better against strong and complex opponents, with VDN achieving a perfect win rate. Adding noise to DQN helps it explore efficiently, resulting in a higher win rate. VDN Mixing consistently produces the highest reward, reflecting the power of collaborative learning.

TABLE V
PERFORMANCE EVALUATION OF TRAINED BLUE AGENT

Method	Opponent Type	Win Rate (Red)	Win Rate (Blue)	Average Reward	
				Red	Blue
Dueling DQN Epsilon	Random Policy	0.0	1.0	-2.2322	4.4912
	Trained Policy	0.0	1.0	0.9860	4.8346
	Final Trained Policy	0.8333±0.1	0.1333±0.1	4.2722	3.5759
Dueling DQN Noise	Random Policy	0.0	1.0	-0.8878	4.9122
	Trained Policy	0.0	1.0	1.7586	4.9438
	Final Trained Policy	0.2333±0.1	0.4±0.1	4.3527	4.5171
VDN Mixing	Random Policy	0.0	1.0	-0.8729	4.8768
	Trained Policy	0.0	1.0	2.5829	4.9884
	Final Trained Policy	0.0	1.0	2.8536	4.8611

For simpler tasks, a standard DQN is sufficient, but in more difficult multi-agent scenarios with skilled opponents, Noisy DQN and VDN Mixing show superior strength and performance.

E. Training Metrics

To further analyze the learning process, we examine the training metrics for each model.

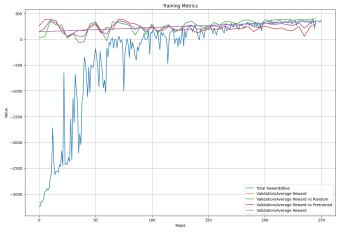

Fig. 2. Training metrics for the Dueling DQN with Epsilon-Greedy

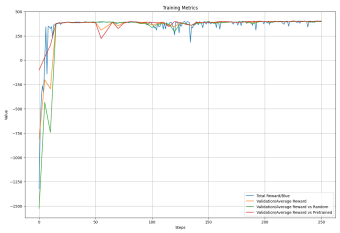

Fig. 3. Training metrics for the Dueling DQN with Noisy Networks

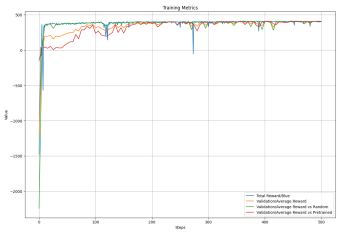

Fig. 4. Training metrics for the Noisy Dueling DQN with VDN mixer

Figures 2, 3, and 4 illustrate the training metrics for the Dueling DQN with Epsilon-Greedy, Dueling DQN with Noisy Networks, and Noisy Dueling DQN with VDN mixer, respectively. These metrics include total reward per episode and validation average rewards against different opponent types.

a) Comparative Analysis of Training Metrics:

- **Learning Stability:** The training curves for the Noisy DQN and VDN models (Figures 3 and 4) exhibit more stable learning compared to the standard DQN with Epsilon-Greedy (Figure 2). This is evident from the smoother increase in total reward per episode and less fluctuation in validation rewards. The Noisy DQN and VDN models show a more consistent upward trend in performance, whereas the standard DQN with Epsilon-Greedy displays more variability.
- **Convergence Speed:** The Noisy DQN and VDN models appear to converge faster towards higher rewards, indicating more efficient learning. This is likely due to the more effective exploration provided by Noisy Networks and the improved coordination enabled by VDN. The standard DQN model shows slower progress and epsilon-greedy exploration might be less efficient in this complex environment.
- **Overall Performance:** The VDN Mixing model achieved the highest reward across all scenarios, demonstrating its superior performance, results, and capabilities in multi-agent contexts. It can be added that the Noisy DQN model also shows us that it improves over the standard DQN, especially in terms of average validation reward. This shows us that both Noisy Networks and VDN contribute to better generalization and better performance in unseen scenarios.

F. Discussion

The results of the above experiments illustrate the efficiency and distinctions of advanced reinforcement learning approaches in solving the problem of MaGent2 Battle which is a complex, multi-agent environment.

- **Model Comparison:** The Noisy Dueling DQN with VDN mixer model outperformed the standard DQN with epsilon-greedy in terms of both win rate and average rewards, especially against the stronger final trained policy. This demonstrates the effectiveness of combining a Dueling architecture, Noisy Networks for exploration, and a VDN mixer for multi-agent coordination.
- **Benefits of Noisy Networks:** The comparison between the standard DQN and the Noisy DQN clearly shows the benefits of integrated exploration. Noisy Networks allow for more stable and efficient learning by introducing stochasticity directly into the network weights, leading to better exploration of the state-action space. The smoother and faster convergence seen in the training metrics further supports this observation.
- **Effectiveness of VDN:** The VDN mixer consistently achieved the best results, proving its ability to effectively decompose the team value function and enable better coordination among agents by dynamically adjusting the

mixing function based on the global state. This adaptability is crucial for handling the complexities of multi-agent interactions and dynamic environments.

- **Advanced Techniques:** The use of the Dueling Q-Network architecture and Prioritized Experience Replay (PER) likely contributed to the overall performance of all models. The Dueling architecture improves value estimation by separating state values and action advantages, while PER focuses learning on the most important experiences. These techniques when combined with Noisy Networks and VDN mixers have resulted in a more efficient learning model

G. Future Work

- Conduct more testing to optimize the values of hyperparameters. Experiment and implement alternative network architectures to better handle complex problems.
- Add more difficult state representation methods, incorporate domain knowledge into model design to enhance learning and application when faced with complex problems.
- Evaluate models with more scenarios in MAgent2 environment to provide a more comprehensive understanding and their adaptability.

V. Conclusion

This study demonstrates the effectiveness of advanced reinforcement learning techniques, such as Noisy Networks, Dueling Q-Networks, and Value Decomposition Networks (VDN), in the complex multi-agent environment of MAgent2 Battle. The Noisy Dueling DQN with VDN mixer model exhibited the most promising results, highlighting the importance of integrated exploration and effective coordination for achieving superior performance in multi-agent settings. The experimental results offer valuable insights into the effectiveness of these methods and suggest that combining these techniques can lead to significant improvements in learning efficiency and stability. Future work will focus on exploring further enhancements to the model architecture, incorporating more sophisticated state representations, and evaluating the models in more diverse scenarios to further improve their robustness and adaptability.

VI. Code Availability

The source code used to compute the results presented in this work is available at: https://github.com/angWindy/Reinforcement-Learning-Final-Project.

References

[1] Mnih, V., Kavukcuoglu, K., Silver, D., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540), 529–533.

[2] Fortunato, M., Azar, M. G., Piot, B., et al. (2018). Noisy networks for exploration. *ICLR*.

[3] Sunehag, P., Lever, G., Gruslys, A., et al. (2018). Value-decomposition networks. *ICLR*.

[4] Rashid, T., Samvelyan, M., de Witt, C. S., et al. (2018). Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. *ICML*.

[5] Wang, Z., Schaul, T., Hessel, M., et al. (2016). Dueling network architectures for deep reinforcement learning. *ICML*.

[6] Schaul, T., Quan, J., Antonoglou, I., & Silver, D. (2016). Prioritized experience replay. *ICLR*.

[7] Zheng, L., Yang, J., Cheng, H., Weng, C., & Shi, Z. (2018). Magent: A many-agent reinforcement learning platform for artificial collective intelligence. *IEEE Access*, 6, 39099-39110.

[8] Tan, M. (1993). Multi-agent reinforcement learning: Independent vs. cooperative agents. *Proceedings of the tenth international conference on machine learning*, 330-337.