21AIE401 – DEEP REINFORCEMENT LEARNING

Predator and Prey Game using Deep Reinforcement Learning

Team Utopia

Group Number – 22

Name	**Registration Number**
C. Charan	BL.EN.U4AIE19012
D. Tharun	BL.EN.U4AIE19016
Sandeep MC	BL.EN.U4AIE19058

Problem Statement:

- To work on the problem, which is a popular multi-agent environment in which multiple agents called predators to capture a prey in a grid world using DQN and MADDPG.

Introduction:

- Two kinds of agents are evolving in the environment. We denote by "prey" by Burglary and "predator" as Police.

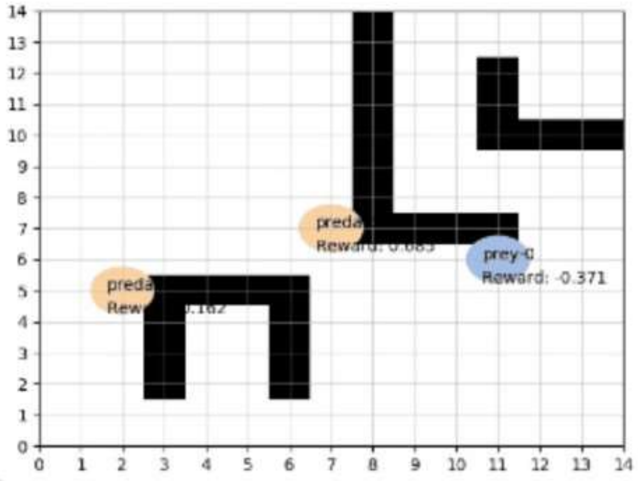
- In the simplest modelisation, the agents are set in a closed square environment where there are obstacles.

- We will be using DRL techniques like Deep Q Network and Multi-Agent Deep Deterministic Policy Gradient.

RL Formulation:

- **Environment**: Our's is a 15 × 15 grid environment, which consists of :

    1). Prey.
    2). Predators.
    3). Obstacles.



RL Formulation:

- **State Space** : The state of the environment includes the locations of all the predators, location of the prey, and Obstacles.

- **Actions**: Move-up, Move-down, Move-left, and Move-right, or Stand-Still.

- **Rewards**: For a predator at position X = (x, y). If Xp = (x_p, y_p) is the position of the prey closest to the predator, we define

$$r_{\text{predator}}(\boldsymbol{X}, \boldsymbol{X}_p) = e^{-c\|\boldsymbol{X}-\boldsymbol{X}_p\|^2}$$

$$r_{\text{prey}}(\boldsymbol{X}, \boldsymbol{X}_p) = 1 - 2e^{-c\|\boldsymbol{X}-\boldsymbol{X}_p\|^2}$$

Literature Survey:

S.NO	Authors names	Title of the Paper	Inference from the paper
1.	Lee KM, Ganapathi Subramanian S, Crowley M.	Investigation of Independent Reinforcement Learning Algorithms in Multi-Agent Environments (2022)	• This paper compares the performance of independent algorithms on four Petting Zoo scenarios that cover the three fundamental types of multi-agent environments .
2.	L. Ang, k. Micheal, Yixin luo.	Deep Reinforcement Learning in Continuous Multi Agent Environments	• In this paper, they assess the effectiveness of algorithms [5] made to operate in both continuous and discrete using (DQN, DDPG, MADDPG).

Literature Survey:

S.NO	Authors name(or) s	Full title of the paper with year	Inference from the paper(based on methodology, technology)
3.	Wang, X.shan, Jun Cheng, and Lei Wang.	Deep-Reinforcement Learning-Based Co-Evolution in a Predator–Prey System (2019).	• In this study [4], they use deep reinforcement learning techniques to give creatures the ability to learn, and then they use the Monte Carlo simulation algorithm to mimic their evolutionary process in a vast habitat.
4.	Y. Lin, Z. Ni and X. Zhong.	Multi-Virtual-Agent Reinforcement Learning for a Stochastic Predator-Prey Grid Environment (2022).	• In this paper [1], they present a novel multi-virtual-agent reinforcement learning (MVARL) approach for a grid-based predator-prey game.

Research Gaps:

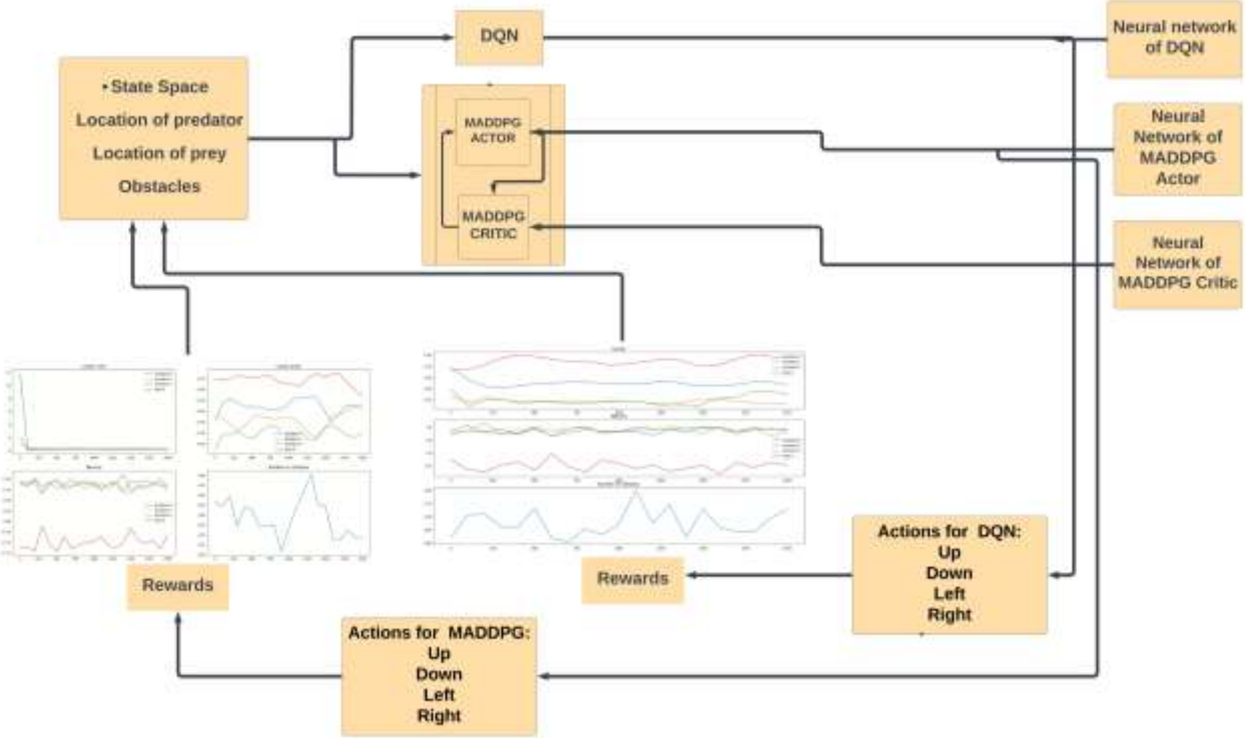
- Many existing studies of predator and prey game using RL have focused on simplified environments, such as grid worlds or continuous state spaces.

- Another research gap is to apply DRL to more complex predator and prey scenarios.

- In some cases, it may be useful to have humans and machines working together to solve predator-prey problems.

- Very few have worked on deep reinforcement learning techniques.

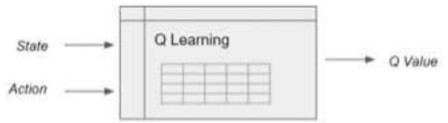
System Architecture:

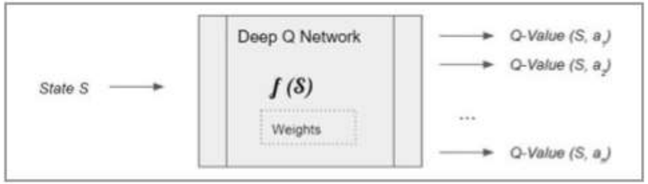


DQN (Deep Q-Network) :

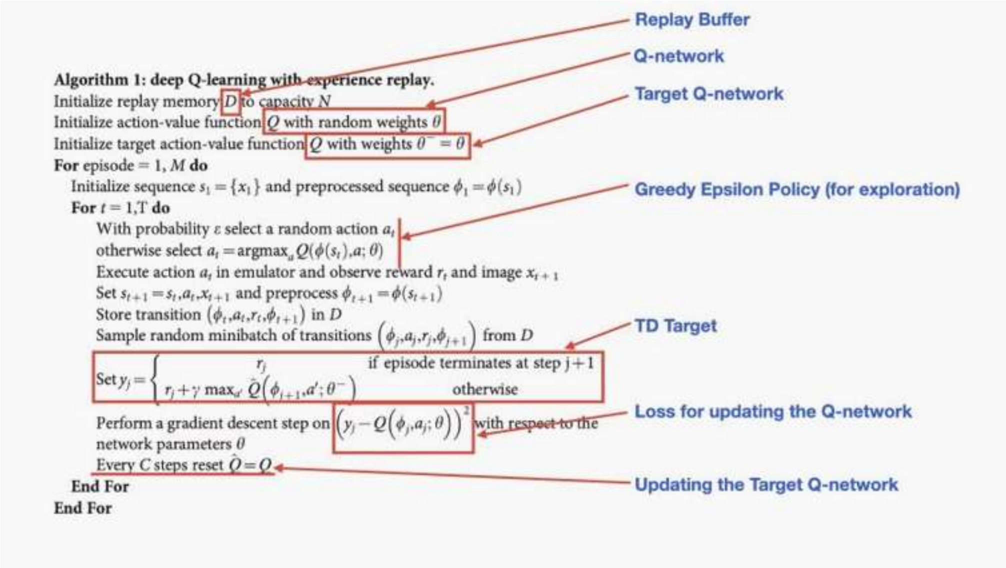
- Deep Q Networks (DQN) are neural networks (and/or related tools) that utilize deep Q learning in order to provide models.

- The algorithm was developed by enhancing a classic RL algorithm called Q-Learning with deep neural networks and a technique called experience replay.





DQN (Deep Q-Network) :

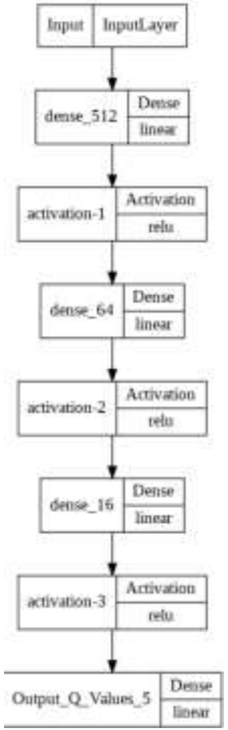


Algorithm 1: deep Q-learning with experience replay.
Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
For episode = 1, M **do**
Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
For t = 1,T **do**
With probability ε select a random action a_t
otherwise select $a_t = \operatorname{argmax}_a Q(\phi(s_t), a; \theta)$
Execute action a_t in emulator and observe reward r_t and image x_{t+1}
Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

$$\text{Set } y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$$

Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ
Every C steps reset $\hat{Q} = Q$
End For
End For

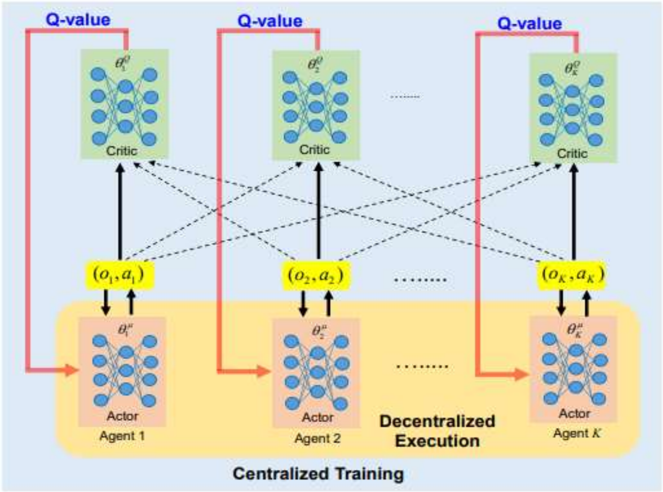
DQN in Our Project Model Architecture:



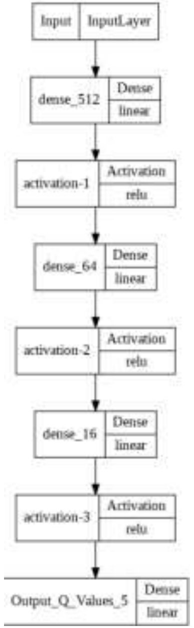
- As a first approximation, we simplify our problem by making the agents independent to one another.
- The DQN neural network design that we have chosen in our project consists of 6 layers.

Multi-Agent Deep Deterministic Policy Gradient:

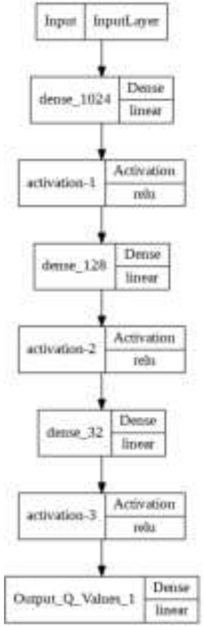
- MADDPG is a multi-agent extension of DDPG that is especially made to deal with continuous action space.



Multi-Agent Deep Deterministic Policy Gradient:



Actor-Network



Critic Network

Implementation Details:

- Environment is custom implemented.
- Matplotlib is used for rendering game.
- Py-Torch is used for implementing DQN and MADDPG agents.
- Trained the agents for about 16000 episodes.

Training:

- The training is done for 16000 episodes.
- DQN - A three-layer neural network is trained with learning rate of 0.9, epsilon is initially kept as 0.9 and then gradually decreased to 0.1 with a epsilon decay of 500000.
- MADDPG - A three-layer neural network for both actor and critic is trained learning rate of 0.9, epsilon is initially kept as 0.9 and then gradually decreased to 0.1 with a epsilon decay of 500000.

Hyper parameters:

- Discount factor
- Learning rate
- Initial epsilon value
- Epsilon decay
- Update frequency

Hyper parameters:

```
gamma: 0.6 # Discount factor.
EPS_START: 0.5 # Probability of exploration.
EPS_END: 0.1
EPS_DECAY: 500000
lr: 0.05 # Learning rate
lr_actor: 0.001 # Learning rate for actor
update_frequency: 25 # Update the target net every...
```

```
gamma: 0.9 # Discount factor.
EPS_START: 0.7 # Probability of exploration.
EPS_END: 0.1
EPS_DECAY: 300000
lr: 0.05 # Learning rate
lr_actor: 0.005 # Learning rate for actor
update_frequency: 30 # Update the target net every...
```

```
gamma: 0.9 # Discount factor.
EPS_START: 0.9 # Probability of exploration.
EPS_END: 0.1
EPS_DECAY: 500000
lr: 0.01 # Learning rate
lr_actor: 0.001 # Learning rate for actor
update_frequency: 20 # Update the target net every...
```

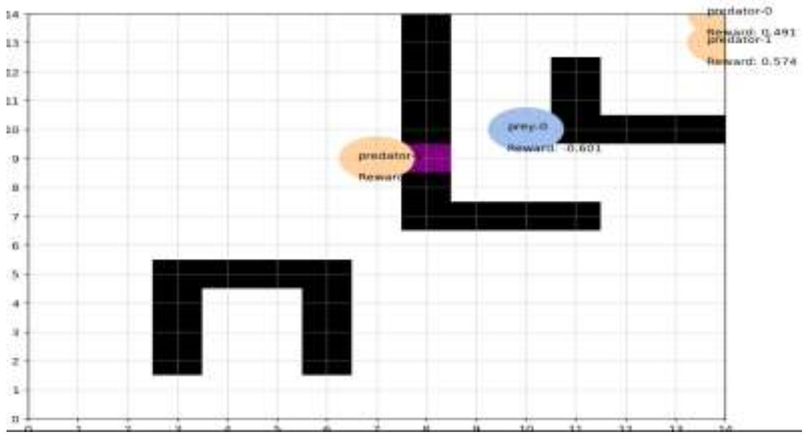
Demo Codes:

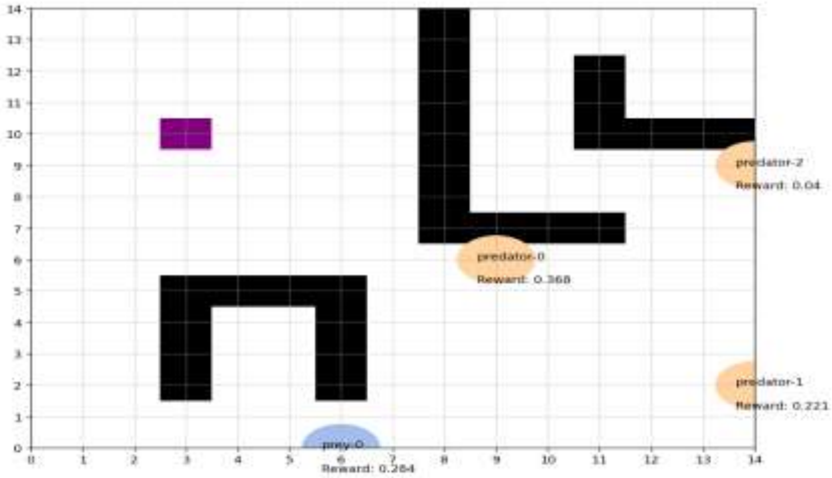
```yaml
agents:
  number_preys: 1
  number_predators: 3
  # For RL
  gamma: 0.9 # Discount factor.
  EPS_START: 0.9 # Probability of exploration.
  EPS_END: 0.1
  EPS_DECAY: 500000
  lr: 0.01 # Learning rate
  lr_actor: 0.001 # Learning rate for actor
  update_frequency: 20 # Update the target net every...
  soft_update_frequency: 50 # Update the target net every..
  update_type: soft # Type of update.
  hidden_size: 32

replay_memory:
  size: 10000 # Maximum size of the memory.
  shuffle: Yes # If Yes, returns random batches among the 
```

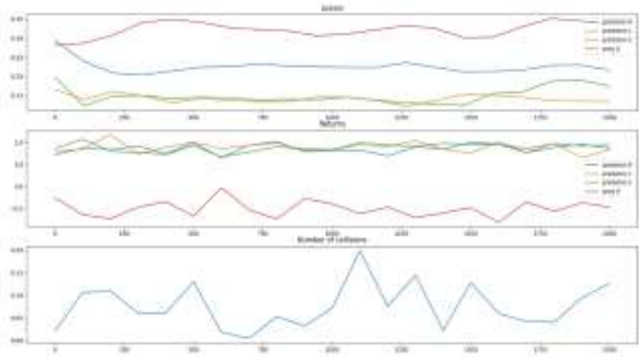
```python
n_actions = 7 if config.env.world_3D else 5
self.n_agents = config.agents.number_preys + config.agents.number_predators
n_obstacles = 2 * len(config.env.obstacles)
self.fc = nn.Sequential(
    nn.Linear(self.n_agents * 3 + n_obstacles + n_magic_switch, 512),
    nn.ReLU(),
    nn.Linear(512, 64),
    nn.ReLU(),
    nn.Linear(64, 16),
    nn.ReLU(),
    nn.Linear(16, n_actions),
)
```

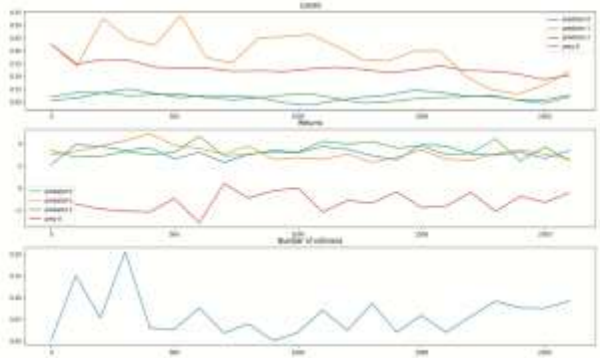
Results:

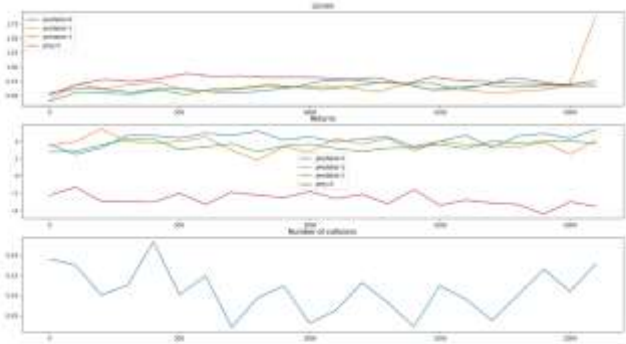

predator-0
Reward: 0.491
predator-1
Reward: 0.574
prey-0
Reward: -0.601


predator-2
Reward: 0.04
predator-0
Reward: 0.368
predator-1
Reward: 0.221
prey-0
Reward: 0.284

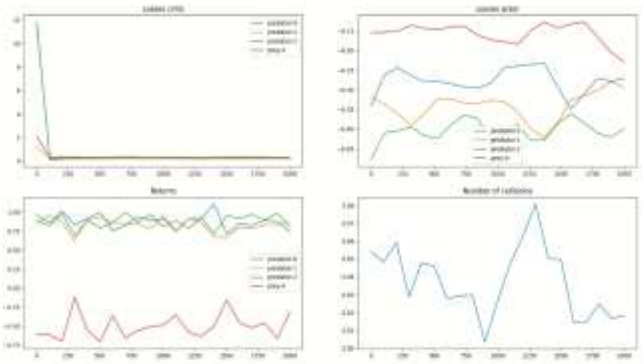
Results for DQN:

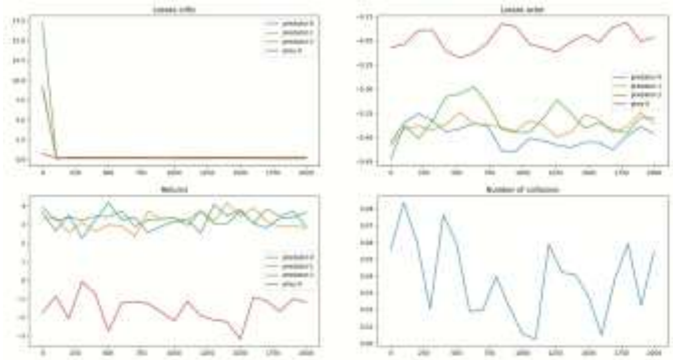


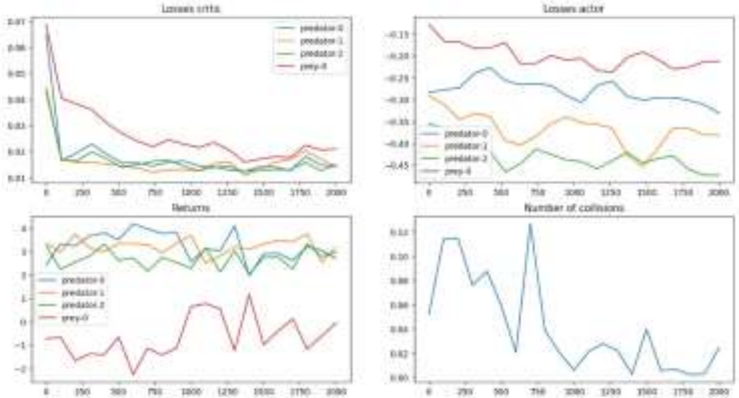




Results for MADDPG:







Conclusion and Future Scope:

- We employ two distinct algorithms: DQN and MADDPG and assessed their performance using a wide range of scenarios.

- Our results with DQN and MADDPG are quite impressive: predators learn to chase prey quickly and well, whereas prey do not learn as quickly as predators.

- We can achieve even better results with a more powerful GPU and more training.

References:

- Y. Lin, Z. Ni and X. Zhong, "Multi-Virtual-Agent Reinforcement Learning for a Stochastic Predator-Prey Grid Environment," 2022 International Joint Conference on Neural Networks (IJCNN), 2022, pp. 1-8.

- Wang, Xueting, Jun Cheng, and Lei Wang. 2019. "Deep-Reinforcement Learning-Based Co-Evolution in a Predator–Prey System" Entropy 21.

- G. Gao and R. Jin, "An End-to-end Flow Control Method Based on DQN," 2022 International Conference on Big Data, Information and Computer Network (BDICN), 2022, pp. 504-507, doi: 10.1109/BDICN55575.2022.00098.

- Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments.

- M. Zhan, J. Chen, C. Du and Y. Duan, "Twin Delayed Multi-Agent Deep Deterministic Policy Gradient," 2021 IEEE International Conference on Progress in Informatics and Computing (PIC), 2021, pp. 48-52.

Thank You