Predator and Prey Game using Deep Reinforcement Learning

1st C.Charan, 2nd D.Sai Tharun Reddy, 3rd Sandeep Preetha M C
[1,2,3] *Department of Computer Science and Engineering*
Amrita School of Computing ,Bengaluru, India
bl.en.u4aie19012@bl.students.amrita.edu,bl.en.u4aie19016@bl.students.amrita.edu,bl.en.u4aie19058@bl.students.amrita.edu

***Abstract*—A range of real-world sectors, including computer network administration, surveillance, and even computer war game simulations, involve patrolling activities. And it is a challenging multi-agent activity to proceed with, so in order for the group as a whole to function at its best, agents must frequently coordinate their decision-making. With the help of deep reinforcement learning techniques, we are able to make the correct decisions and move to the desired location. In this project, we are tackling the issue of a well-known multi-agent environment in which numerous predator agents use deep reinforcement learning techniques like Deep-Q Network (DQN) and Multi-Agent Deep Deterministic Policy Gradient (MADDPG) to catch one or more prey in a grid-based environment.**

***Index Terms*—Patrolling activities, Multi-agent environment, Deep Reinforcement Learning, DQN, Multi-Agent Deep Deterministic Policy Gradient(MADDPG).**

I. Introduction

The term ”patrol” comes from the verb ”patrol,” which means ”to walk or wander about an area at regular intervals, to safeguard or monitor it.” The ability to efficiently perform this patrolling activity has potential utility in a variety of application areas where dispersed surveillance, inspection, or control is necessary. This is seen in many different types of games and in many different types of real-world situations, including business competition and war. We're making a predator-prey game for our project. The Predator and Prey Game is made up of two different kinds of agents that are constantly changing in the environment; these agents are referred to as prey and predator respectively. We refer to the police as the ”predator” and burglars as the ”prey” in our terminology.

Through playing this game, we hope to instill in the predators the ability to track and capture their prey (s). In this context, which features both cooperative and competitive elements, multiple agents may be present on either side, and the objective of each individual agent is to maximize its own benefit. In the most basic form of this model, the agents are placed in an environment consisting of four square walls. There are also obstructions in this environment, and each actor has the same observation of it, which is the position of every other agent. Deep reinforcement learning, also known as DRL, has a lot of untapped promise when it comes to tackling difficult decision-making issues in predator-prey games, particularly in multi-agent settings. By combining reinforcement learning and deep neural networks at scale, DQN and MADDPG were able to solve a broad variety of Atari games, some of which were beyond human capability. Deep Q Networks and MADDPG are not specific neural network architectures; rather, they are a generic term that can refer to a variety of neural network structures, such as convolutional neural networks and other structures that use specific methods to learn about a variety of processes. They have been used in a number of artificial intelligence (AI) applications, including image recognition, speech recognition, and translation.

II. Literature Survey

In recent years, playing a variety of uncertain strategy games has seen significant application of reinforcement learning. Card games, Predator and Prey, go, and other games are among them .Without any prior human experience, reinforcement learning has been demonstrated to be capable of superhuman performance in strategic games. Additionally, a variety of works have already been published. In this paper [1], they present a novel multi-virtual-agent reinforcement learning (MVARL) approach for a grid-based predator-prey game. While taking in information from these virtual agents, a global agent also engages with the real world. This method improves the generalization capabilities of reinforcement learning in dynamic environments while also lowering the overall computational cost.

This paper [2] compares the performance of independent algorithms on four Petting Zoo scenarios that cover the three fundamental types of multi-agent environments: cooperative, competitive, and mixed. They demonstrate that, when parameter sharing is combined with the inclusion of agent indicators, independent algorithms outperform a number of multi-agent algorithms, including multi-agent proximal policy optimization and multi-agent deep deterministic policy gradient, in fully observable environments. In this work, QMIX and Counterfactual Multi-Agent Policy Gradients, two algorithms that particularly address the multi-agent credit-assignment problem, are examined (COMA).

In order to lessen traffic congestion, this research [3] tries to apply deep reinforcement learning algorithms for signal control at a specific intersection. The sequence and length

of signal phases are suggested to be changed using a deep deterministic policy gradient (DDPG)-based method in order to reduce the average vehicle delay time. The outcomes demonstrate that in terms of efficiency and stability, the DDPG algorithm beats the DQN algorithm and the fixed-time control strategy. Because it combines a value-based method with a policy-based method and uses the actor-critic mechanism in its algorithm, DDPG outperforms DQN. While DDPG can output continuous action, DQN can only output discrete action.

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.As a result, the suggested model shows how a learning mechanism affects a predator-prey ecosystem and shows how AI approaches may be used to predict how behavior would evolve in a predator-prey ecosystem.

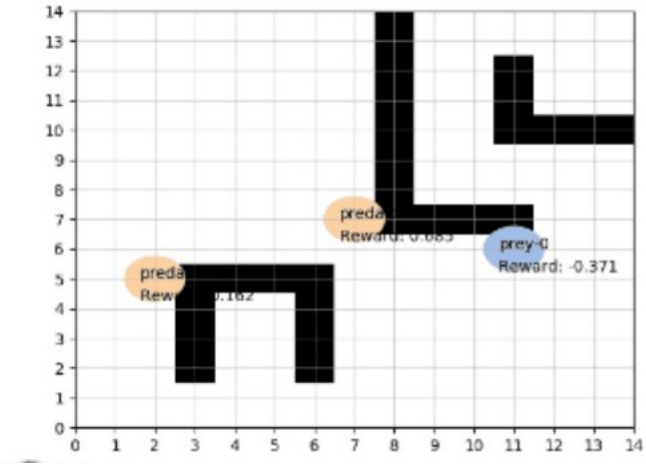
In this effort, they assess the effectiveness of algorithms [5] made to operate in both continuous and discrete (DQN) action spaces (DDPG, MADDPG). They make advantage of the OpenAI Gym framework's multi-agent extension to build our predator-prey habitat [1]. This framework is employed because it is rapidly emerging as the industry norm for testing settings for reinforcement learning algorithms.The results from our DQN network are extremely impressive (which was unexpected). They should be more adept at developing cooperative and competitive strategies than DQN agents in multi-agent environments, in addition to being competitive themselves.

III. RL Formulation

In this section, we shall talk about the RL formulation and the assumptions made in order to achieve the expected reward and further succeed in the implementation. The MDP formulation is generally represented by a tuple with three parameters (s, a, r), namely the current state, the action performed on that state, the reward obtained, and the environment. The key purpose of an MDP requires the understanding and definition of four major components, as seen below: the environment, states, actions, and reward system.//

A. Environment:

In a real-world scenario, the selected problem statement has a vast environment. But on further narrowing down for a faster computation, we developed a custom environment for the predator and prey in 2D and further assigned obstacles to increase the complexity of the agents. The environment consists of multiple agents (predators and preys), and the agents work in collaboration to maximize the cumulative return in a competitive setting. The agents are placed in a closed square (2-D) environment with obstacles placed at random locations.



B. State Space:

We define the state space as S = [0, 1] power of 3N where N is the number of agents in the environment. Therefore, S S represents the normalized 3D coordinates of the N agents. In our simulations, we consider the z coordinate is fixed to 0 and the agents are forced into the 2D plane. We can therefore consider S = (1, 2, · · ·, Ns) power of 3N, where Ns is the number of steps we use in our discretization. For our experimentation, we consider discretization of 15 steps along x and y axes results in a total 225 possible states in the entire 2-D environment for an individual agent. The state of the environment takes into account the actual position of all of the predators as well as the locations of the prey.

C. Action Space:

In this environment the both predator and prey have the privilege to perform 5 different actions to catch or escape from one-another respectively. The actions that can be performed are denoted by A, where A = Stay, front, left, right, back. Here, the stay function means the agent doesn't move from its current state. However, as many actions the agents can perform, they are not allowed to move outside the environment and thereby they will be given a negative reward, thereby the environment forces the agent to stay within the dedicated 2-D environment.

1). Move-Up,
2). Move-Down,
3). Move-Left,
4). Move-Right or
5). Stand-Still.

D. Reward Function'

Reward is the most integral part of the RL formulation to obtain an effectively trained agent. After numerous trial and error of reward functions, ri : S ×A × S → R where i 1, · · · , N represents the agent and ri(s, a, si) is the reward given to agent for the transition (s, si) done with action a.

We further normalize the obtained results for the next state between the range [-1,1]. For further elucidating the reward function separately for both predator and prey, let us consider the position of predator to be X = (x, y) and the predators position is Xp = (xp,yp,zp)n a 2 D coordinate.

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

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

Where Constant c = 5 in our experiment. The reward is calculated based on the distance between the two agents. We use the exponential function to evaluate because as the agents move far away they will be assigned with heavy negative rewards, which will result in the predator to move closer and catch the prey and simultaneously prey tries to move far away from the predator (as a consequence the rewards for prey is designed to have one minus the value of the predator's reward.

IV. FORMALIZATION

A. Reinforcement Learning

An agent will continue to iterate through the RL process by transitioning from one state to the next up to the point where it achieves the end state, also known as the goal state. As a result of the agent's actions in its surroundings, the agent is rewarded or punished accordingly. The objective of the agent is to accumulate as many rewards as possible. These concepts can be presented in a more formal manner by using the Markov decision process (S, A, R, P, and γ).

The first term is "S," and it represents all of the many states that are possible. "A," on the other hand, represents all of the different actions that the agent is capable of performing in its current state. The value "R" represents the breakdown of the possible rewards. The transition probability distribution is determined by the value "P," and the amount that the future rewards should be discounted is established by the discount factor (γ).

The agent will have to deal with a lot of unknowns, hence it's necessary to use this hyper-parameter. In the RL algorithm, the agent prefers to take the actions that result in the greatest sum of rewards. The end goal for the agent is to find the best possible course of action (). The course of action that ought to be performed in all and all situations is laid forth in a policy. RL algorithms are designed to learn an optimal policy that maximizes the projected cumulative discounted reward and maps states to the best feasible actions in order to achieve this goals.

B. Decision Points:

- First Decision Point: Selecting Value Function: Within the scope of our project, the approximation of the action-value function estimate is denoted by the letter Q. (s, a).

$$Q^*(s,a) = \mathbb{E}_{s'\sim p}[R(s,a,s') + \gamma \max_{a'} Q^*(s',a')]$$

Estimates of Q can then be specified using the weights of a neural network, a state, and an action.

- Second Decision Point: Selecting Neural Network architecture - the architecture we are using is state in action value-out architecture, the state variables for our application are the locations of our agents which represents the present state of our grid which is also our environment and action variables are move the agents up, down, left, right and stand still.
- Third Decision Point: Selecting what to optimize - Our project's goal is to earn a high reward for the number of episodes. We must minimize the loss w.r.t to optimal action value function.
- Fourth Decision Point: Selecting the targets for policy evaluation - in our application we use off policy TD target for policy evaluation.
- Fifth Decision Point: Choosing an exploration strategy - In our project, we used the epsilon-greedy method throughout agent training. Epsilon is initially kept as 0.9 and then gradually decreased to 0.1 with a decay factor of 0.09 and after every 10000 steps.
- Sixth decision point: Selecting a loss function – We used MSE (Mean Squared Error) as our loss function.
- Seventh decision point: Selecting an optimization method – we used Adam as an optimiser.

C. Deep Q-Network

Deep Q Networks, also known as DQNs, are related tools for neural networks that make use of deep Q learning. These networks are used to provide models, such as simulations of intelligent video game play. Some researchers have proposed that the name "Deep Q Networks" doesn't refer to a particular type of neural network design, but rather to a collection of neural network architectures that employ various techniques to gain understanding of various processes. Deep neural networks and a technique called experience replay were brought together in this program in an effort to make standard reinforcement learning (RL) algorithm Q-Learning more effective.

The fact that DQN is off-policy facilitates the implementation of a Replay Buffer. Each time the agent interacts with its surroundings, the interaction is recorded in the buffer as a tuple which contain four parameters consisting of the following: 'state, next state,action,reward". For the neural network to learn new things, the gradient updates are calculated at each every time step using a sample size of previous events that are taken from the pool equally and at random. This is done on a very basic level by collecting samples, storing those samples, and using those samples for experience replay to update the Q network. It does not make use of a Q-table but rather a Neural Network that, given

a state, estimates the Q-values associated with each action based on that state. In order to acquire knowledge of policies based on high-dimensional sensory input, the technique of deep Q learning often makes use of something that is referred to as general policy iteration. General policy iteration can be regarded as a result of the combination of policy evaluation and policy iteration.

The utilization of a conventional RL strategy becomes laborious and time-consuming due to the state space's dynamic nature and size, which is the primary motivation behind the decision to adopt deep reinforcement learning for this technique. In addition, there are several agents involved in the challenge, and there is more than one predator and prey in the grid. Making use of Deep RL helps to simplify the grid, which in turn leads to improved and more accurate performance. The arguments presented below demonstrate why DQN is more effective than conventional RL's Q-learning.

i. When it comes to approximating the Q function, DQN relies on multi-layer neural networks rather than the tabular Q function that is utilized in Q-learning. In other words, the action-value function Q(.,.) that is used in DQN is matched by Q(.,.; θ(t)), where denotes the parameters of the neural network. This makes it possible to perfect the functioning of the neural network.

ii. Rather than using one the Q function for calculating both target and estimation, DQN uses two separate neural networks, which are a target neural network and an online neural network Q(·, ·;). More specifically, for any experience ,When calculating the target, DQN makes use of the target network, and when calculating the estimation,Q(St,Ut,θ(t)), it makes use of the online network. After that, the goal and estimation values are utilized in the computation of the loss, which is as follows:

$$[r_t + \gamma \max_{u_{t+1}} \bar{Q}(s_{t+1}, u_{t+1}; \bar{\theta}_t) - Q(s_t, u_t; \theta_t)]^2$$

We can conceptualize the DNN as a black box .As input, it accepts the current state of the game and outputs an approximation of the Q value for each of the available actions. After that, we select the course of action that yields the highest Q-value, just like Q-learning does. But in contrast to that, we are now attempting to identify patterns rather than mapping each state to the action that is most appropriate for it. This wouldn't be conceivable in settings that have large state spaces to begin with. In order for our neural network to make predictions based on its surroundings, we need to provide it with matched sets of input and output. The neural network will learn from those data to provide an approximation of the output depending on the input by changing the parameters in an iterative fashion.

Therefore, below are the stages of the reinforcement learning process that make use of deep Q-learning networks (DQNs):

1. Memory is where the individual keeps a record of all of their previous experiences.

2. The following step will be done based on the Q-maximum network's output.

3. The loss function in this scenario is the mean squared inaccuracy of the predicted Q-value compared to the intended Q-value minus Q*.This is essentially a problem with going backwards. Due to the fact that we are working with an issue involving reinforcement learning, we are not aware of either the objective or the actual value. Getting back to the equation that was obtained from the Bellman equation in order to update the Q-value. we have:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right]$$

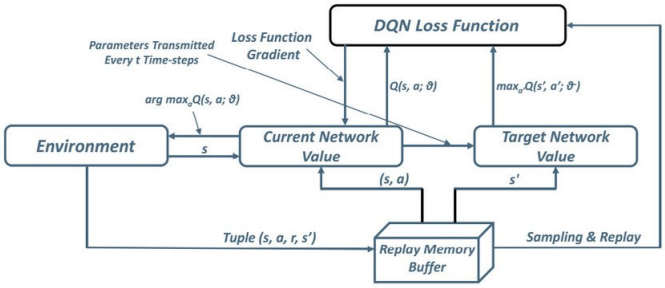


Fig. 1. Diagrammatic Representation of the DQN's Working Principle.

The target is denoted by the portion that is colored green. We could argue that it is anticipating its own value, but since R is the truly unbiased reward, the network will adjust its gradient using backpropagation in order to eventually converge on a value.

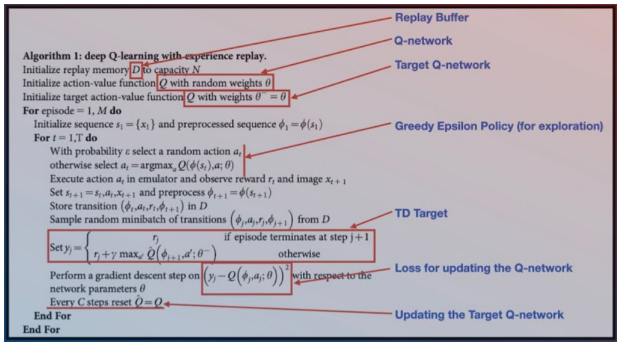


Fig. 2. Algorithm for Deep-Q Network With Experience Replay Method.

D. Multi Agent Deep Deterministic Policy Gradient (MADDPG):

MADDPG is a multi-agent extension of DDPG designed specifically for dealing with continuous action space. We

will attempt to implement MADDPG, which employs policy gradient and actor/critic configuration, in order to eliminate the assumption that each agent's Q-function is independent of the actions of the other agents. This contains two distinct neural networks (Actor and critic). In our project, the state space is the agent's position, with the environment, obstacles, and other factors also considered. This is what an actor network does. The actions (stay, forward, left, right, and back) that the actor believes are optimal given the provided condition will be output. The critic network's job is to evaluate both the state and the action space as input from the actor network and to give the value of the specific action chosen by the actor network, as well as to decide the cumulative reward based on whether the action performed by the actor is a good or bad action. The goal of each agent is to learn the Q-function (critic) as well as the policy (actor).

We present a training method that uses an ensemble of policies for each agent, resulting in more reliable multi-agent policies and the learning of a unique centralized critic for each agent as we consider a multi-agent scenario. We sampled the actions for the exploration policy using the Gumbel-SoftMax function distribution because the environment we evaluated is a discrete action space. Because each agent's reward is highly dependent on the actions of other agents, policy gradient approaches have the drawback of being challenging to train. As a result, the drawbacks may be substantial.

The Actor Critic algorithm is made to avoid the drawbacks of REINFORCE and other Policy Gradient Algorithms. It uses two neural networks: an actor that controls how our agent behaves and a critic that assesses how well the action taken is based on values (policy-based). We update at every stage as opposed to waiting until the end of the episode, as in Monte Carlo REINFORCE. Additionally, we are unable to use the overall rewards because we update at each time step. Instead, we must develop a critic model that closely resembles the value function.

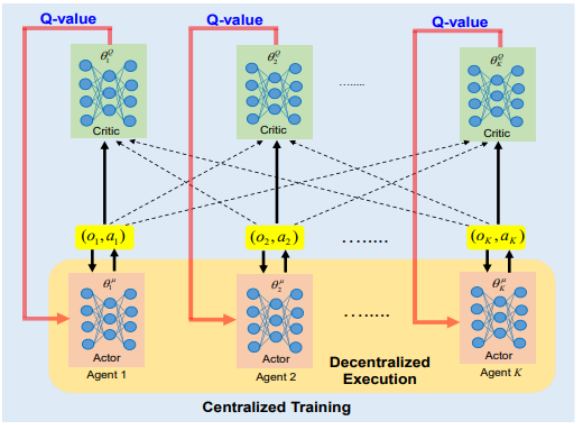


Fig. 3. Centralized training and decentralized execution framework in MADDPG algorithm.

The agents receive additional information, such as observations and activities of other agents, in addition to local observations of the environment. By doing this, repetition is avoided, and it also prevents the state space from becoming too complicated and predators from receiving their ideal reward. We will therefore develop a central critic system that maintains the states and behaviors of each actor network.

for *each agent* $k \in \mathcal{K}$ **do**
  Initialize replay buffer $\mathcal{D}_k$.
  Initialize the actor network $\mu_k\left(o_k|\theta_k^{\mu}\right)$ and critic network $Q_k\left(s_k, a|\theta_k^{Q}\right)$ with weights θ_k^{μ} and θ_k^{Q}, respectively.
  Initialize the target networks, μ_k' and Q_k' with weights $\theta_k^{\mu'}$ and $\theta_k^{Q'}$, respectively.
end
for *each episode* $e = 1, 2, \ldots$ **do**
  for *each agent* $k \in \mathcal{K}$ **do**
    Initialize random process $\mathcal{N}_k$ for exploration.
    Generate initial local observation from the environment simulator.
  end
  for *each step* $t = 1, 2, \ldots$ **do**
    for *each agent* $k \in \mathcal{K}$ **do**
      Select action $a_k\left(t\right) = \mu_k\left(o_k\left(t\right)|\theta_k^{\mu}\right) + \mathcal{N}_k\left(t\right)$
    end
    Execute joint action $a\left(t\right) = \left(a_1\left(t\right), \ldots., a_K\left(t\right)\right)$
    for *each agent* $k \in \mathcal{K}$ **do**
      Collect reward $r_k\left(t\right)$ and observe $s_k\left(t+1\right)$.
      Store the transition $\left(s_k\left(t\right), a\left(t\right), r_k\left(t\right), s_k\left(t+1\right)\right)$ into $\mathcal{D}_k$.
      Sample random minibatch of B transitions $\left(s_k^i, a^i, r_k^i, s_k^{i+1}\right)$ from $\mathcal{D}_k$.
      Update the critic network by minimizing the loss given by (19).
      Update the actor policy using the sampled policy gradient given by (20).
      Update the target networks according to (21).
    end
  end
end

Fig. 4. Algorithm for MADDPG algorithm.

V. System Architecture:

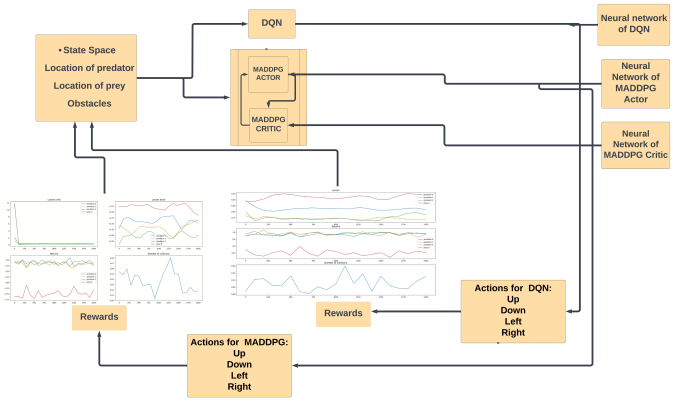


Fig. 5. System Architecture.

Initially, we take neural network inputs from our state space, which contains the locations of both agents and obstacles. In

our project, we will be using two DRL models, which are DQN and MADDPG. For DQN, we will have one neural network, but for the MADDPG, we need to have two different neural networks for both the actor and the critic. The first layer in the DQN model takes input from state space, which is the locations of our agents and obstacles, and after traversing the entire network, it returns an output of size 5, which are the Q-Values of our action space. After receiving the action values, they are returned to state space, where they can receive rewards, and this loop will continue indefinitely. The MADDPG model actor will take inputs from state space and give the output to the critic model. Along with the output from the actor, the critic also takes the input values from state space and gives the best action values; these action values are sent to the actor, and after we send the action values to perform and be able to get rewards, the same loop iterates for n number of episodes.

A. Deep Q Network (DQN):

We reduce our problem as a first approximation by making the agents independent of one another. Each agent has a unique Q function, which depends only on its state and behavior. The DQN neural network design that we have chosen in our project consists of 6 layers in which three are dense layers and other three are ReLU activation functions.First layer takes the input from state space, which are locations of our agents, and after going through the whole network, it gives an output of size 5, which are Q-Values of our action space.

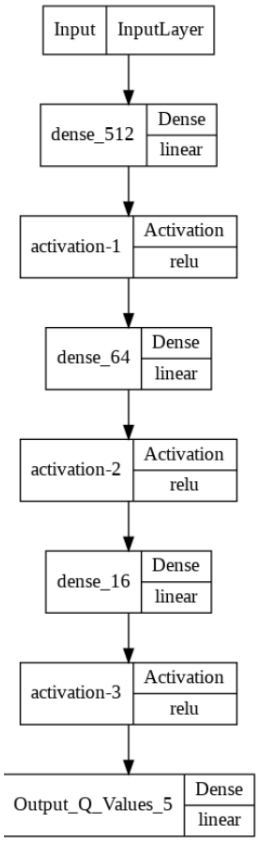


Fig. 6. DQN's Neural Network Architecture.

B. Multi-Agent Deep Deterministic Policy Gradient (MADDPG) :

In MADDPG, we uses an actor/critic setting which uses both a policy learning method and Q-Learning.So, we will have two different neural network architecture for both actor and critic. In our model the actor neural network design consists of 6 layers, in which three are dense layers and other three are ReLU activation functions First layer takes the input from state space, which are locations of our agents, it gives an output of size 5, which are Q-Values of our action space. And the critic neural network design in our model also consists of 6 layers, which are same as in actor, here also it will take input from state space along with actions form actor and gives an output of size 1, which is the best Q-Value of our action space.

VI. Training:

The training has been completed for almost 16000 episodes. The training procedure makes use of hyper parameters such as discount factor,learning rate ,an initial epsilon value, epsilon decay, update frequency to update the target network.We apply an epsilon greedy exploration policy of 0.9 at start, then declining to 0.1 after 10000 learning step. The learning rate is set to 0.01 and the frequency of updating the target net (hard update) is set to once per 20 learning steps. The replay memory buffer has a maximum size of 10000 elements, and we shuffle batches among them. We additionally introduce noise

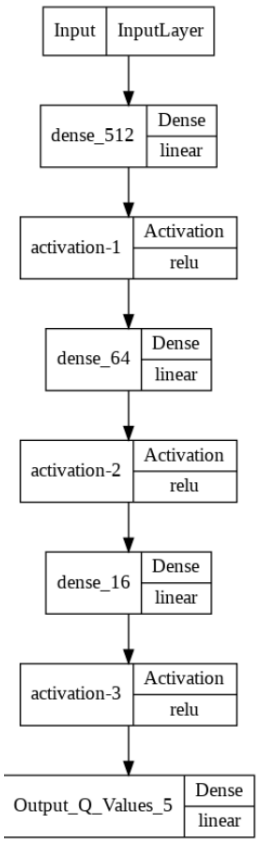


Fig. 7. Actor Neural Network Architecture.

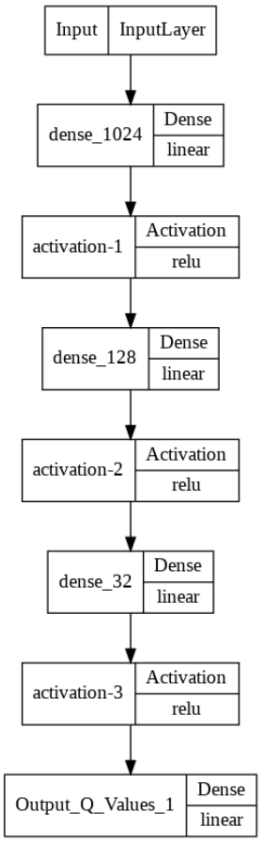


Fig. 8. Critic Neural Network Architecture.

to the agents' behaviors, and they fail with a chance of 0.1 percent.For learning, we utilize a batch size of 200. We plot the mean of the reward (averaged over 100 episodes) and the loss averaged on 100 episodes.

A. *Hyper-parameter Tuning*

Hyper parameter tuning is performed on 5 such as discount factor, learning rate, an initial epsilon value, epsilon decay, update frequency to update the target network. For the purpose of tuning, we have considered five hyper parameters which include two different values of discount factor (0.6, 0.9), three different values of initial epsilon value (0.5,0.7,0.9), two different values of learning rate (0.01,0.05), two different values of epsilon decay (300000,500000), and three different values of update frequency (20,25,300). we found that the optimal values for hyper parameters for high rewards are as follows: discount factor of 0.9, a learning rate of 0.01, an initial epsilon value of 0.9, epsilon decay value of 500000, and update frequency value of 20.

```
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...
```

Fig. 9. Values of Hyperparameters-1.

```
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...
```

Fig. 10. Values of Hyperparameters-2.

```
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...
```

Fig. 11. Values of Hyperparameters-3.

VII. Results and Discussion:

In figures 11,12 and 13 the first graph box with blue, green, orange lines indicate the losses for predators, and red line indicate the losses for prey, whereas the second graph box indicates the rewards for respective agents as mentioned above. And finally the last graph box indicates the mean of number of times all the agents are getting collide with the obstacles for over 2000 episodes for DQN model. And after comparing the all the three hyper-parameter values, the third hyper-parameter are getting good results compared to others.

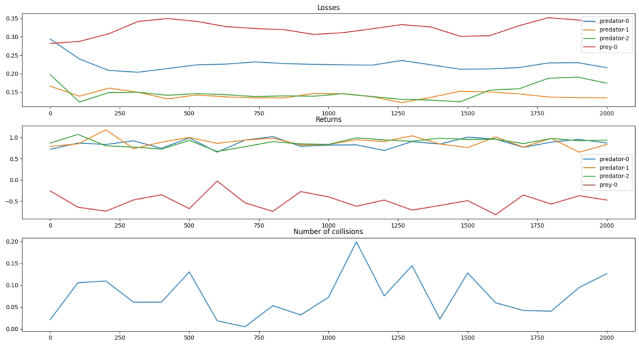


Fig. 12. Reward, Loses graph values of Hyperparameters-1 for DQN.

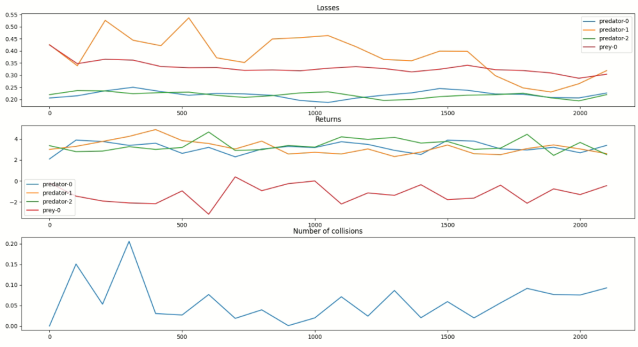


Fig. 13. Reward, Loses graph values of Hyperparameters-2 for DQN.

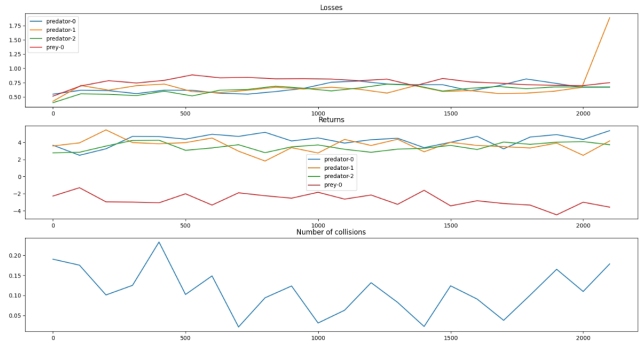


Fig. 14. Reward, Loses graph values of Hyperparameters-3 for DQN.

In figures 14,15 and 16 the first graph box with blue, green, orange lines indicate the losses for predators, and red line indicate the losses for prey for critic, whereas the second graph box also indicates the losses for agents respectively for actor and the third graph indicates the rewards for respective agents for both predators and prey mentioned above. And finally the last graph box indicates the mean of number of times all the agents are getting collide with the obstacles for over 2000 episodes for MADDPG model. And after comparing the all the three hyper-parameter values, the third hyper-parameter are getting good results compared to others.

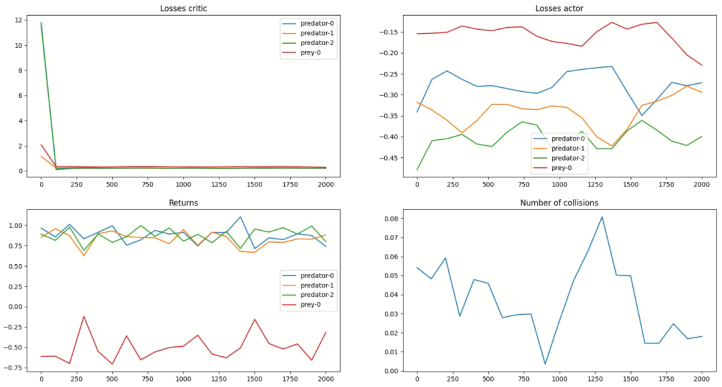


Fig. 15. Reward, Loses graph values of Hyperparameters-1 for MADDPG.

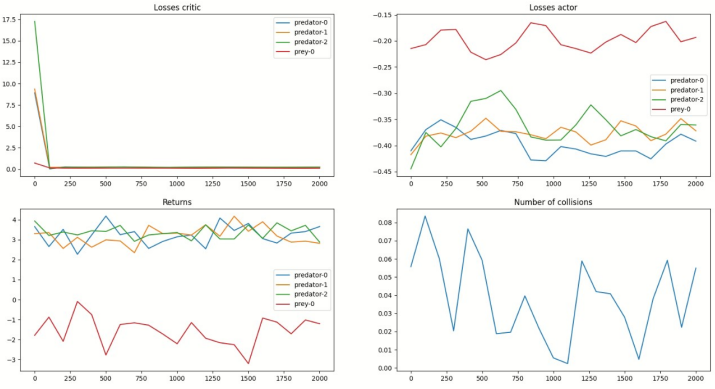


Fig. 16. Reward, Loses graph values of Hyperparameters-2 for MADDPG.

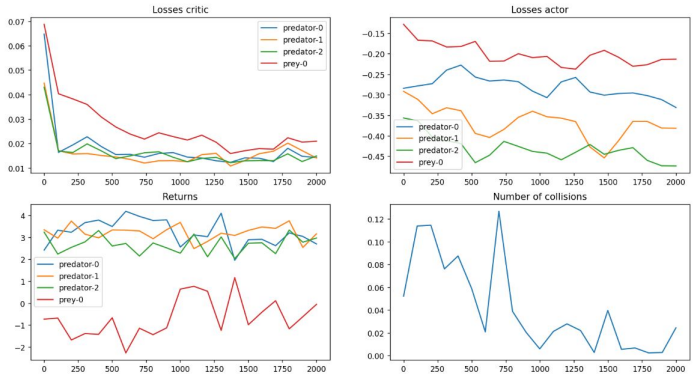


Fig. 17. Reward, Loses graph values of Hyperparameters-3 for MADDPG.

VIII. Conclusion and Futher Scope:

We coded a full grid environment with agents and obstacles for this project's implementation. 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. The results for MADDPG and DQN are very similar. However, MADDPG results are slightly higher than DQN. When using MADDPG, it takes so much longer to run episodes compared to the DQN model. More work is required to assist predators in learning.This can be accomplished by speeding them up or changing the reward function, for example. We can achieve even better results with a more powerful GPU and more training.

References

[1] 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.

[2] Lee KM, Ganapathi Subramanian S, Crowley M. Investigation of independent reinforcement learning algorithms in multi-agent environments. Front Artif Intell. 2022 Sep 20;5:805823. doi: 10.3389/frai.2022.805823. PMID: 36204598; PMCID: PMC9530713.

[3] H. Pang and W. Gao, "Deep Deterministic Policy Gradient for Traffic Signal Control of Single Intersection," 2019 Chinese Control And Decision Conference (CCDC), 2019, pp. 5861-5866, doi: 10.1109/CCDC.2019.8832406.

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

[5] L. Ang, k. Micheal, Yixin luo, S. Rohan, "Deep Reinforcement Learning in Continuous Multi Agent Environments", 2020 Chinese Control And Decision Conference (CCDC), 2020.

[6] 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.

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

[8] J. Park Lee, J. Kim, A. Taehwan, I. Jooyoung. “ Co-Evolution of Predator-Prey Ecosystems by Reinforcement Learning Agents” . Entropy. 23. 461. 10.3390/e23040461.

[9] Wang, X., Cheng, J. and Wang, L., 2019. Deep-reinforcement learning-based co-evolution in a predator–prey system. Entropy, 21(8), p.773.

[10] 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, doi: 10.1109/PIC53636.2021.9687069..