In the name of women, life, freedom

Fatemeh Salehi – 810198423

Project 3

Getting Familiar with the game's environment

Windy Grid world is a standard grid world with start and goal states. The difference is that a crosswind runs upward through the middle of the grid. Actions are the standard four: up, right, down, and left. In the middle region, the resultant next states are shifted upward by the "wind" whose strength varies from column to column. The reward is -1 until the goal state is reached.

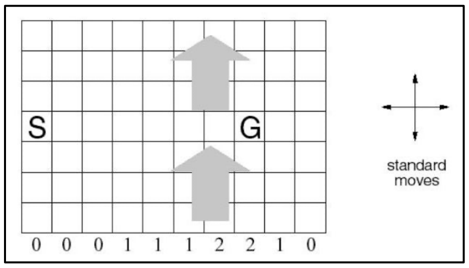


Fig 1: windy grid world

Q-Learning

Q-learning is a model-free, off-policy reinforcement learning that will find the best course of action, given the agent's current state. Depending on where the agent is in the environment, it will decide the next action to be taken.

Step 1: Create an initial Q-Table with all values initialized to 0 and an epsilon initialized to 1, which will degrade at a constant rate.

Step 2: If epsilon is less than 0.05 (or some other threshold) choose a random action and perform it unless choosing the action with the highest value. Update values in the table.

Step 3: Get the value of the reward and calculate the value Q-Value using Bellman Equation

Step 4: Continue the same until an episode ends

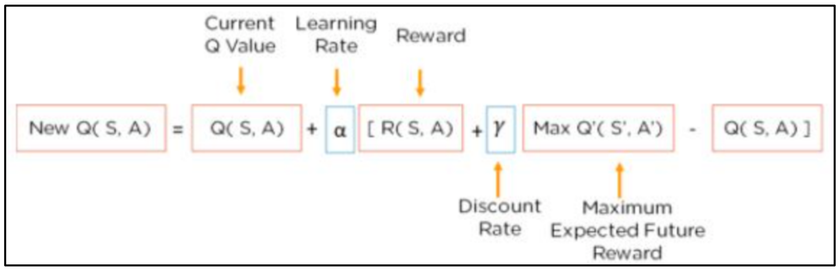

Current Q Value
Learning Rate
Reward
New Q(S, A) = Q(S, A) + α [R(S, A) + γ Max Q'(S', A') - Q(S, A)]
Discount Rate
Maximum Expected Future Reward

Fig 2: Bellman Equation

We ran the code for 3 different values of the discount factor and the results are as follows:

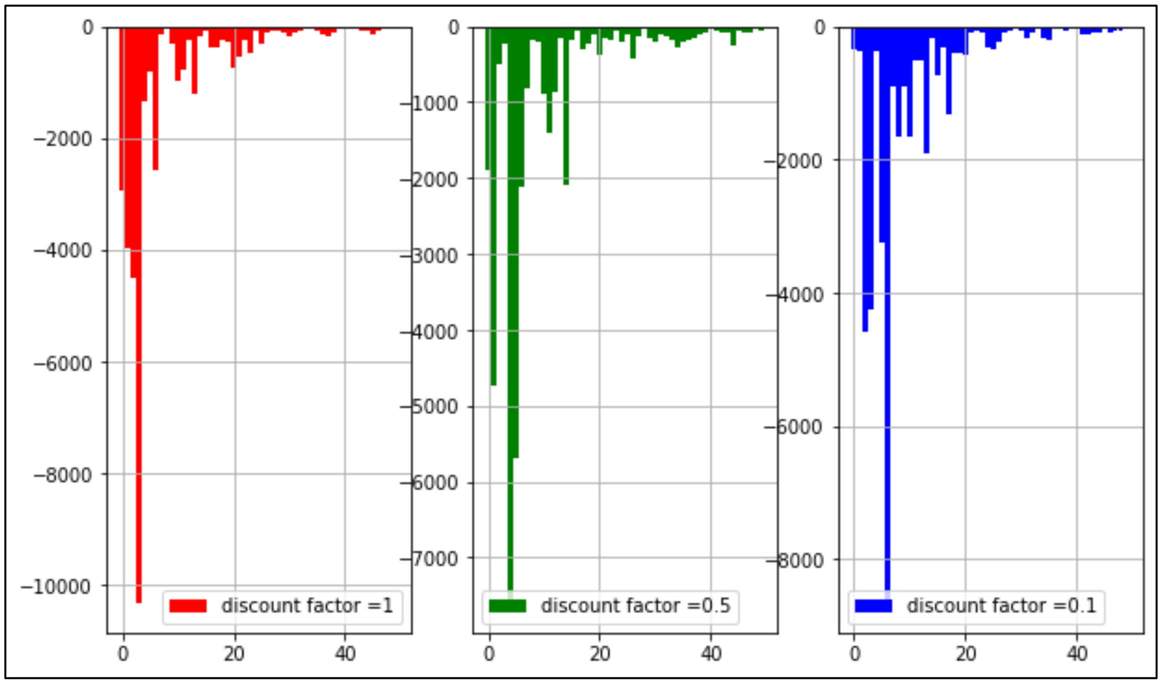


Fig 3: the rewards in each episode of Q-learning for different discount values

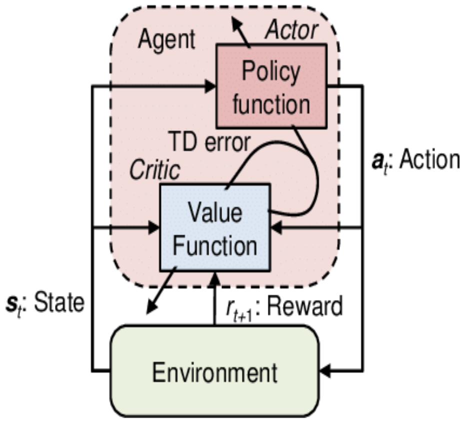
According to Fig 3, the greater the discount factor, the faster the convergence rate; it is worth mentioning that for smaller discount factors, the Q-values are stuck in a local optimal solution, which makes the overall reward not change over a period.

Actor-Critic

In this algorithm, the temporal difference is defined by value function

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

$V(s_{t+1}) \rightarrow$ The value of the next state may give more insight about how the last action was good rather than blending this value to the action taken in the next step.



$$softmax\ policy: \pi_t(s,a) = \Pr\{a_t = a | s_t = s\} = \frac{e^{P(s,a)}}{\sum_b e^{P(s,b)}}$$

$$P(s_t, a_t) \longleftarrow P(s_t, a_t) + \beta\delta_t$$

$$V_{t+1}(s_t) = V_t(s_t) + \alpha\delta_t = V_t(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$$

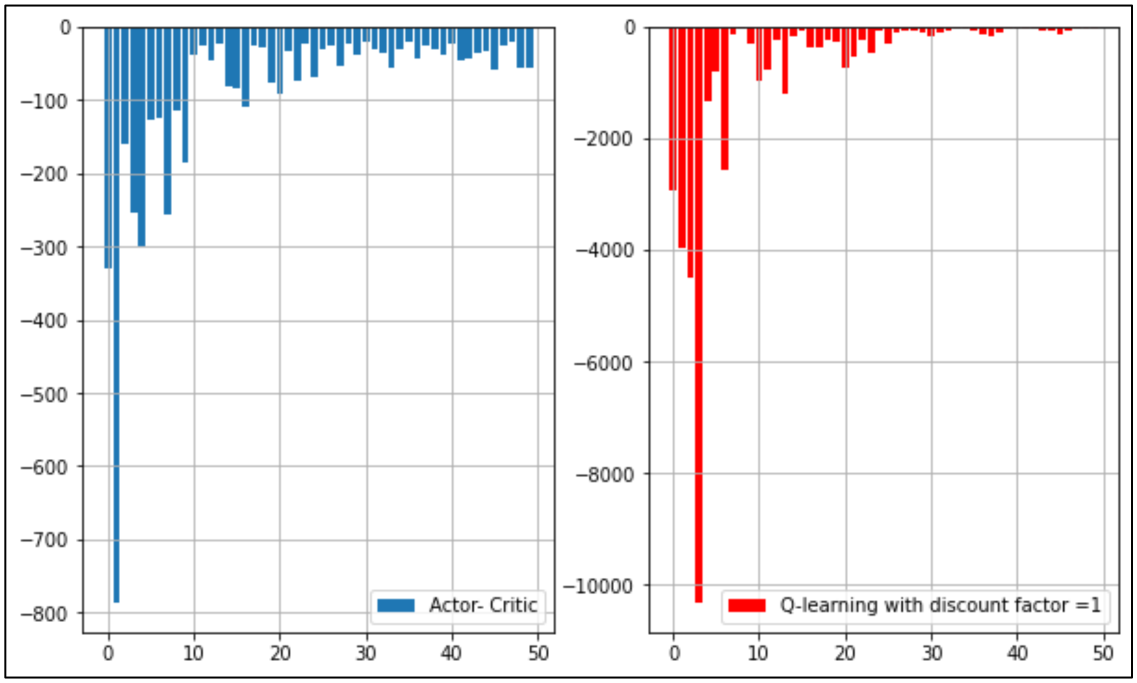
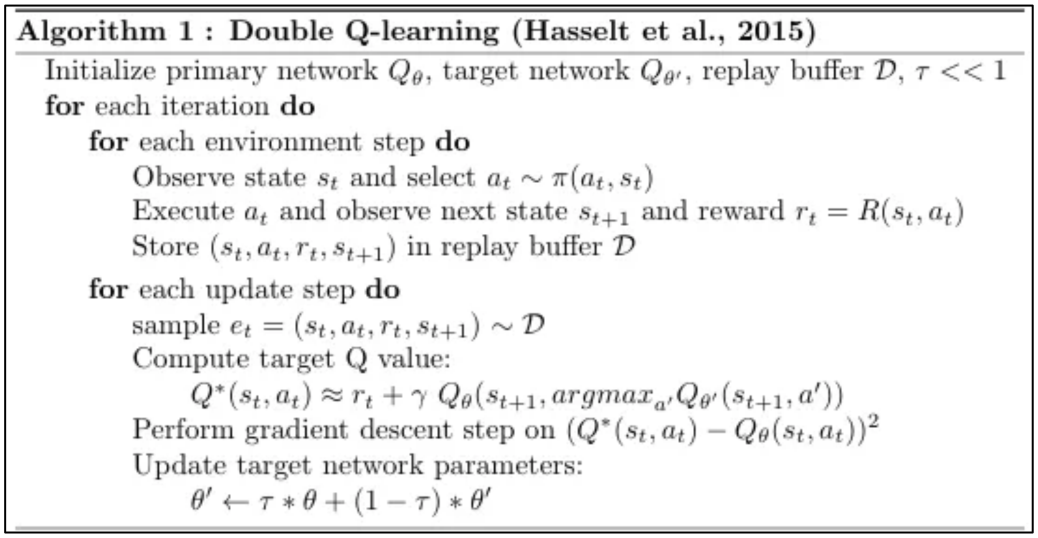


Fig 4: The rewards in each episode of Q-learning with a discount factor of 1 and Actor-Critic

According to Fig 4, we can conclude that Q-learning works better than Actor-Critic, which means that the action that the agent takes in the next state is critical, and only state value is not enough to evaluate a state.

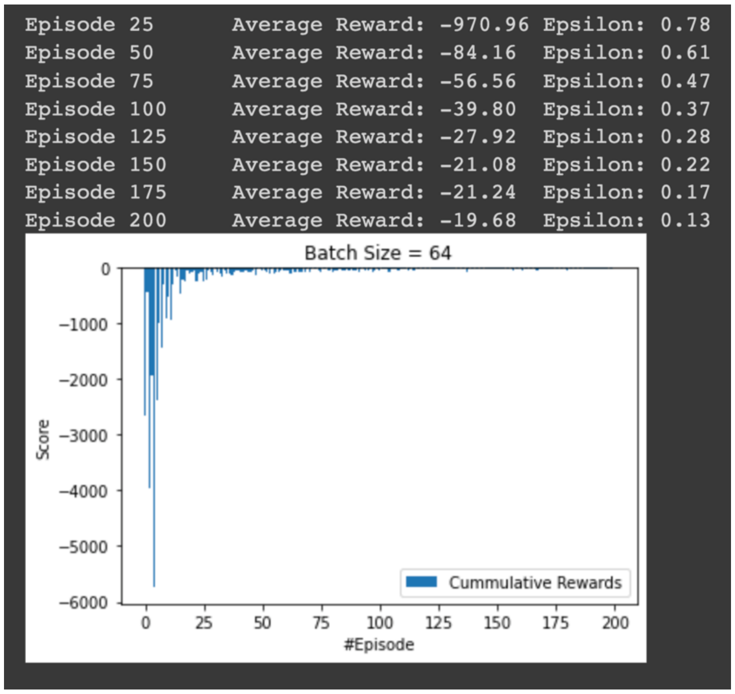
DDQN (Double Deep Q-Network)

This method involves using two separate Q-value estimators, each of which is used to update the other. Using these independent estimators, we can unbiased Q-value estimates of the actions selected using the opposite estimator [1]. We can thus avoid maximization bias by disentangling our updates from biased estimates.



```
Algorithm 1 : Double Q-learning (Hasselt et al., 2015)
  Initialize primary network $Q_\theta$, target network $Q_{\theta'}$, replay buffer $\mathcal{D}$, $\tau << 1$
  for each iteration do
      for each environment step do
          Observe state $s_t$ and select $a_t \sim \pi(a_t, s_t)$
          Execute $a_t$ and observe next state $s_{t+1}$ and reward $r_t = R(s_t, a_t)$
          Store $(s_t, a_t, r_t, s_{t+1})$ in replay buffer $\mathcal{D}$
      for each update step do
          sample $e_t = (s_t, a_t, r_t, s_{t+1}) \sim \mathcal{D}$
          Compute target Q value:
              $Q^*(s_t, a_t) \approx r_t + \gamma\ Q_\theta(s_{t+1}, argmax_{a'} Q_{\theta'}(s_{t+1}, a'))$
          Perform gradient descent step on $(Q^*(s_t, a_t) - Q_\theta(s_t, a_t))^2$
          Update target network parameters:
              $\theta' \leftarrow \tau * \theta + (1 - \tau) * \theta'$
```

The output of this algorithm for batch-size of 64 is as following:



```
Episode 25      Average Reward: -970.96 Epsilon: 0.78
Episode 50      Average Reward: -84.16  Epsilon: 0.61
Episode 75      Average Reward: -56.56  Epsilon: 0.47
Episode 100     Average Reward: -39.80  Epsilon: 0.37
Episode 125     Average Reward: -27.92  Epsilon: 0.28
Episode 150     Average Reward: -21.08  Epsilon: 0.22
Episode 175     Average Reward: -21.24  Epsilon: 0.17
Episode 200     Average Reward: -19.68  Epsilon: 0.13
```

Fig 5: The rewards in each episode of DDQN with batch size = 64

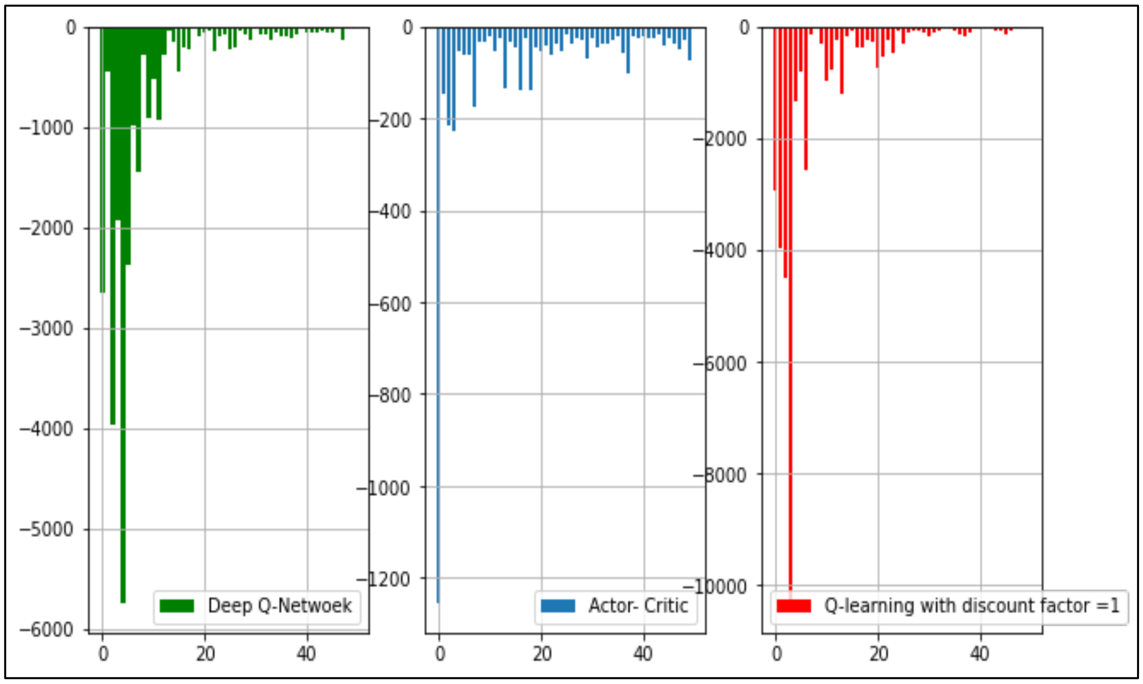
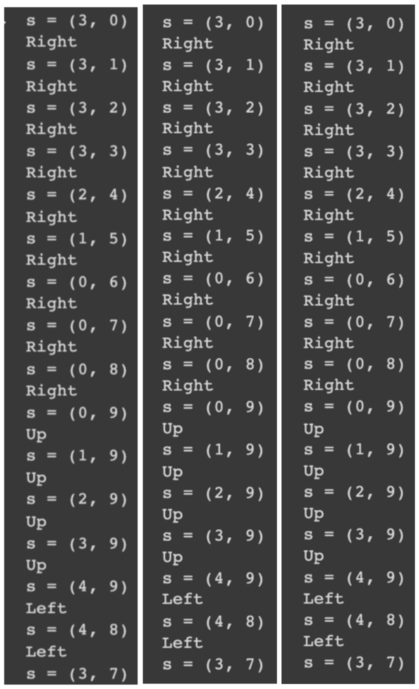


Fig 6: The rewards in each episode of Q-learning with a discount factor of 1, Actor-Critic, and DDQN

According to Fig 6, DDQN converges sooner than both Q-learning and Actor-Critic even though it is more complex than the other two.

Compare the taken paths



```
s = (3, 0)
Right
s = (3, 1)
Right
s = (3, 2)
Right
s = (3, 3)
Right
s = (2, 4)
Right
s = (1, 5)
Right
s = (0, 6)
Right
s = (0, 7)
Right
s = (0, 8)
Right
s = (0, 9)
Up
s = (1, 9)
Up
s = (2, 9)
Up
s = (3, 9)
Up
s = (4, 9)
Left
s = (4, 8)
Left
s = (3, 7)
```

```
s = (3, 0)
Right
s = (3, 1)
Right
s = (3, 2)
Right
s = (3, 3)
Right
s = (2, 4)
Right
s = (1, 5)
Right
s = (0, 6)
Right
s = (0, 7)
Right
s = (0, 8)
Right
s = (0, 9)
Up
s = (1, 9)
Up
s = (2, 9)
Up
s = (3, 9)
Up
s = (4, 9)
Left
s = (4, 8)
Left
s = (3, 7)
```

```
s = (3, 0)
Right
s = (3, 1)
Right
s = (3, 2)
Right
s = (3, 3)
Right
s = (2, 4)
Right
s = (1, 5)
Right
s = (0, 6)
Right
s = (0, 7)
Right
s = (0, 8)
Right
s = (0, 9)
Up
s = (1, 9)
Up
s = (2, 9)
Up
s = (3, 9)
Up
s = (4, 9)
Left
s = (4, 8)
Left
s = (3, 7)
```

Fig 6: The paths taken by Q-learning, Actor-Critic, and DDQN respectively

Multi-agent RL for two agents in the windy grid world

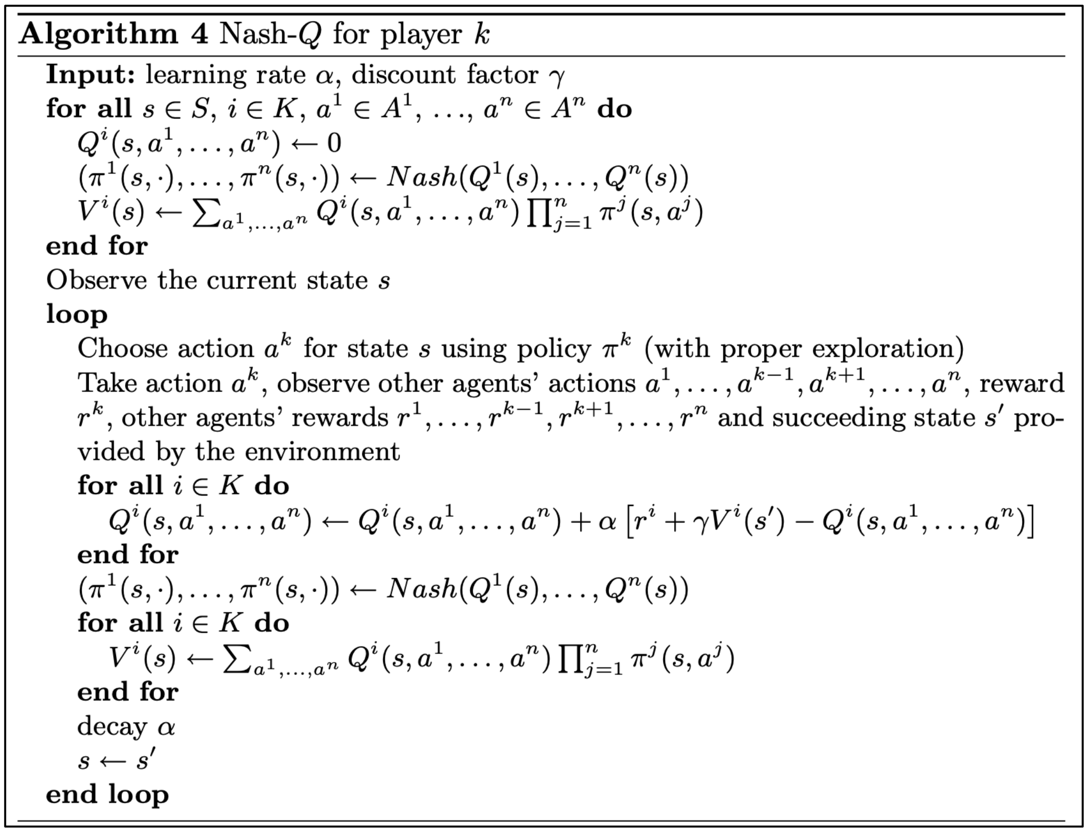
In the environment, two agents will search for a way to the goal position without bumping into each other. The reward for each action is -1; if the agents hit each other the reward would be -10, and if agents are both at the green position, they get the reward of 10.

The Friend-or-Feo Q-Learning

In windy grid world, the agents are friends meaning that they want to help each other out in order to get to the goal.

$$V_i(s) \leftarrow \max_{\substack{a_1 \in A_1 \\ a_2 \in A_2}} Q_i(s, a_1, a_2) \quad i = 1,2$$

$$Q_i(s, a_1, a_2) \leftarrow Q_i(s, a_1, a_2) + \alpha[r + \gamma V_i(s') - Q_i(s, a_1, a_2)] \quad i = 1,2$$



```
Algorithm 4 Nash-$Q$ for player $k$
  Input: learning rate $\alpha$, discount factor $\gamma$
  for all $s \in S$, $i \in K$, $a^1 \in A^1$, ..., $a^n \in A^n$ do
    $Q^i(s, a^1, \dots, a^n) \leftarrow 0$
    $(\pi^1(s,\cdot), \dots, \pi^n(s,\cdot)) \leftarrow Nash(Q^1(s), \dots, Q^n(s))$
    $V^i(s) \leftarrow \sum_{a^1,\dots,a^n} Q^i(s, a^1, \dots, a^n) \prod_{j=1}^{n} \pi^j(s, a^j)$
  end for
  Observe the current state $s$
  loop
    Choose action $a^k$ for state $s$ using policy $\pi^k$ (with proper exploration)
    Take action $a^k$, observe other agents' actions $a^1, \dots, a^{k-1}, a^{k+1}, \dots, a^n$, reward
    $r^k$, other agents' rewards $r^1, \dots, r^{k-1}, r^{k+1}, \dots, r^n$ and succeeding state $s'$ pro-
    vided by the environment
    for all $i \in K$ do
      $Q^i(s, a^1, \dots, a^n) \leftarrow Q^i(s, a^1, \dots, a^n) + \alpha\left[r^i + \gamma V^i(s') - Q^i(s, a^1, \dots, a^n)\right]$
    end for
    $(\pi^1(s,\cdot), \dots, \pi^n(s,\cdot)) \leftarrow Nash(Q^1(s), \dots, Q^n(s))$
    for all $i \in K$ do
      $V^i(s) \leftarrow \sum_{a^1,\dots,a^n} Q^i(s, a^1, \dots, a^n) \prod_{j=1}^{n} \pi^j(s, a^j)$
    end for
    decay $\alpha$
    $s \leftarrow s'$
  end loop
```

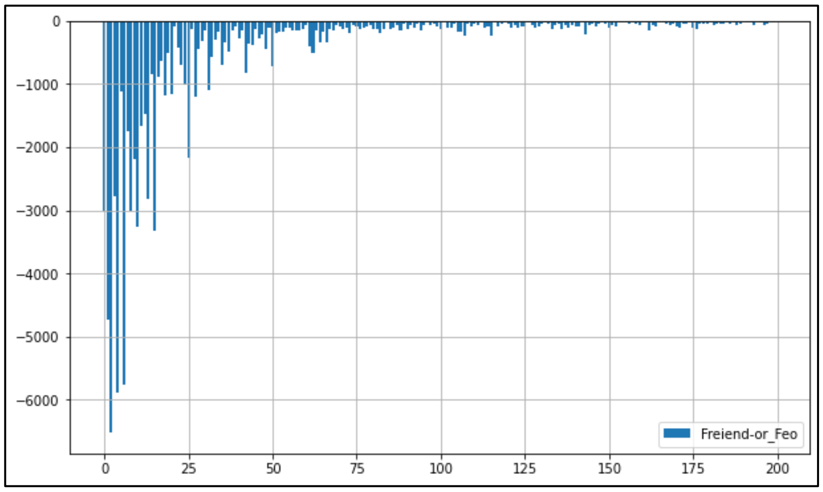


Fig 7: The rewards in each episode of Friend-or-Feo method

The Distributed Q-learning

The idea is that each agent ignores the fact that other agents exist and only consider its local Q-table. The local Q-values are updated only when the update leads to an increase in the Q-value.

$$Q_{i,k+1}\left(x_k, u_{i,k}\right) = \max\left\{Q_{i,k}\left(x_k, u_{i,k}\right), r_{k+1} + \gamma \max_{u_i} Q_{i,k}\left(x_{k+1}, u_{i,k}\right)\right.$$

The local policy is updated only if the update leads to an improvement in the Q-value.

$$\bar{h}_{i,k+1}(x_k) = \begin{cases} u_{i,k} & if \max\limits_{u_i} Q_{i,k+1}(x_k, u_i) > \max\limits_{u_i} Q_{i,k}(x_k, u_i) \\ \bar{h}_{i,k}(x_k) & otherwise \end{cases}$$

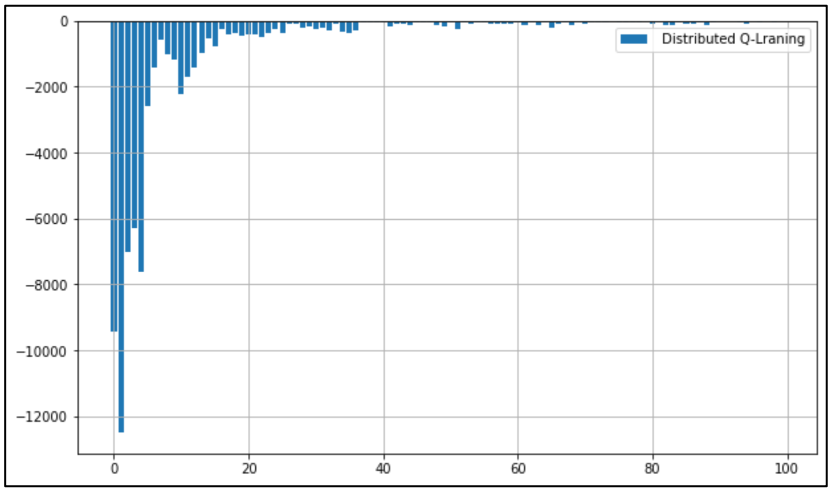


Fig 8: The rewards in each episode of Distributed Q-learning method

Fully Cooperative Method

The agents have the same reward function and the same Q-table; the learning goal is to maximize the common discount return.

$$Q_{k+1}(x_k, u_k) = Q_k(x_k, u_k) + \alpha\left[r_{k+1} + \gamma \max_{u'} Q_k(x_{k+1}, u') - Q_k(x_k, u_k)\right]$$

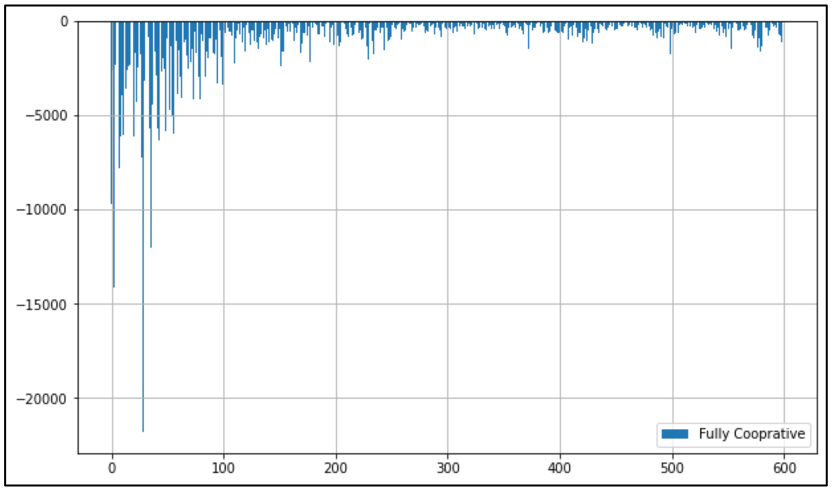


Fig 9: The rewards in each episode of Fully Cooperative method

Conclusion:

According to the Fig7, 8, and 9 the best method for this environment is Distributed Q-learning because it converges way sooner than the other methods.

References:

[1] “Deep Reinforcement Learning with Double Q-learning” (Hasselt et al., 2015).