INTRODUCTION TO MACHINE LEARNING

ASSIGNMENT 3

Submitted to: Dr. Alina Vereshchaka

CSE 574 - INTRODUCTION TO MACHINE LEARNING

May 2023

Jeevalkant Dandona (50485395)
Nihal Mishra (50480980)

University at Buffalo
Graduate School of Education

TABLE OF CONTENTS

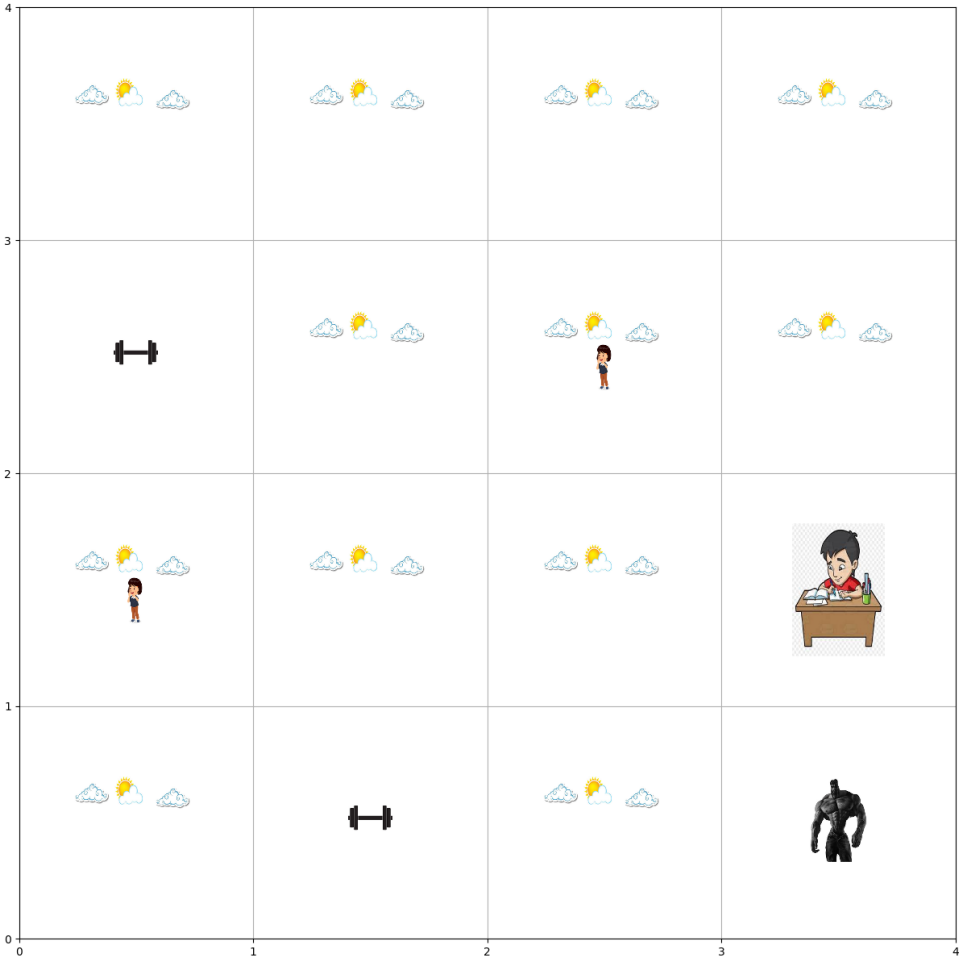
Part-1

Defined Environment

The environment is named Loveworld , It's a 4*4 grid environment.
In this environment a boy moves in 4 directions, left, right up, and down, There are 4 rewarding possibilities if he meets a girls she breaks his heart (-ve reward), if he goes to the gym (+ve reward), if he studies (+ve reward), and finally the goal is to reach the Giga Chad point where he become ***knowledgeable as well as physically fit***(max reward point).

Actions - Left, Right, Up, Down
States - 16 states
Rewards - [Girl (-10), Gym (+10), Study (+25), Giga Chad (+200)]

Environment Visualization



Safety in AI

We monitor the safety of the environment by defining safety constraints that the RL environment must follow, to ensure required outcomes. Monitoring the behavior of the environment, testing on various simulations.

To ensure the agent chooses only allowed actions, constraints and filters can be implemented at the application level, where the model is deployed. These constraints can restrict or filter certain actions that are not deemed safe or appropriate.

Part-2 and Part 3

Tabular Method used to solve problems.

SARSA (STATE-ACTION-REWARD-STATE-ACTION)

Update Function: $Q(s, a) \leftarrow Q(s, a) + \alpha * (r + \gamma * Q(s', a') - Q(s, a))$

SARSA updates Q-values by considering the action taken in the next state according to the current policy.

Advantages

- SARSA is a model-free algorithm, meaning it doesn't need a predefined model of the environment to learn.
- It can directly learn from raw sensory inputs and map them to appropriate actions.
- Despite the complexity of the environment, SARSA has the ability to converge to an optimal policy, ensuring effective decision-making.

Disadvantages

- SARSA may exhibit slow convergence, particularly when dealing with large environments.
- The choice of hyperparameters, such as the exploration rate and learning rate, can significantly impact its performance and sensitivity.
- In certain environments, SARSA may not be able to find the optimal policy, potentially leading to suboptimal results.

Q-Learning

Update Function: $Q(s, a) \leftarrow Q(s, a) + \alpha * (r + \gamma * \max(Q(s', a')) - Q(s, a))$

Q-Learning uses an exploration-exploitation strategy by selecting actions based on an ε-greedy policy.

Advantages

- Q learning is also model free
- Handles Large States Spaces
- Works with less rewards

Disadvantages

- Convergence speed
- Finding the right balance can be challenging.
- May struggle with continuous spaces.

After Applying the Algorithms

SARSA (STATE-ACTION-REWARD-STATE-ACTION)

```
class SarsaAgent:

    def __init__(self, grid, alpha=0.1, gamma=0.99, epsilon=0.98, epsilon_decay=0.009, episodes=100):
        self.dict_q1 = {}
        self.alpha = alpha
        self.gamma = gamma
        self.epsilon = epsilon
        self.epsilon_decay = epsilon_decay
        self.episodes = episodes
        self.stateMap = grid
        self.arr = []
        for state in grid.values():
            self.dict_q1[state] = np.zeros(4)



    def choose_nextaction(self,q_values, epsilon=0.01):
        # actions = [Action("up", 0), Action("down", 1), Action("left", 2), Action("right", 3)]

        actions = [0, 1, 2, 3]
        if(np.any(np.random.rand() < epsilon)):
            selected_action = random.choice(actions)
        else:
            selected_action = np.argmax(q_values)
        return selected_action
    def Q_value_calculation(self, reward, new_state, dict_q1, action, old_state):
        new_state = grid.get(tuple(new_state["position"]))
        q_old = dict_q1[old_state][int(action)]
        new_action = self.choose_nextaction(dict_q1[new_state])
        q_new_action = dict_q1[new_state][int(new_action)]
        dict_q1[old_state][int(action)] += self.alpha * (reward + (self.gamma * q_new_action) - q_old)
        return dict_q1[old_state][int(action)]
```

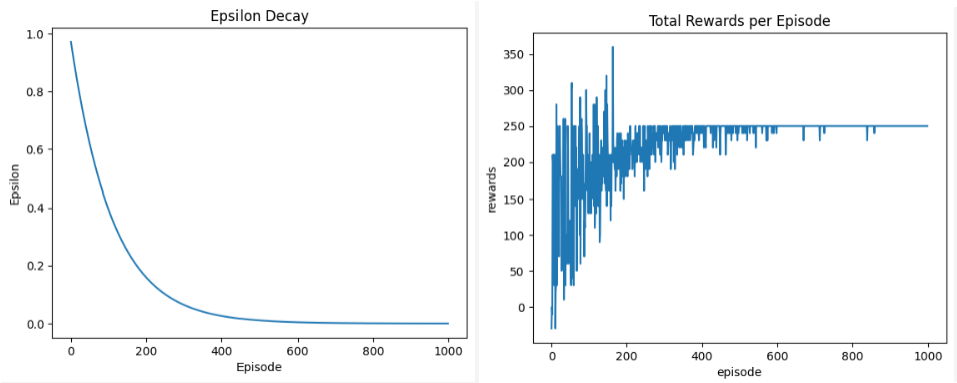
```
grid = {
    (0, 0): 0, (0, 1): 1, (0, 2): 2, (0, 3): 3,
    (1, 0): 4, (1, 1): 5, (1, 2): 6, (1, 3): 7,
    (2, 0): 8, (2, 1): 9, (2, 2): 10, (2, 3): 11,
    (3, 0): 12, (3, 1): 13, (3, 2): 14, (3, 3): 15
}

sarsa_agent=SarsaAgent(grid=grid, alpha=0.1, gamma=0.99, epsilon=0.98, epsilon_decay=0.009, episodes=1000)

arr, final_res = sarsa_agent.train()

plt.plot(arr)
plt.title('Epsilon Decay')
plt.ylabel('Epsilon')
plt.xlabel('Episode')
plt.show()

plt.plot(final_res)
plt.title('Total Rewards per Episode')
plt.ylabel('rewards')
plt.xlabel('episode')
plt.show()
```



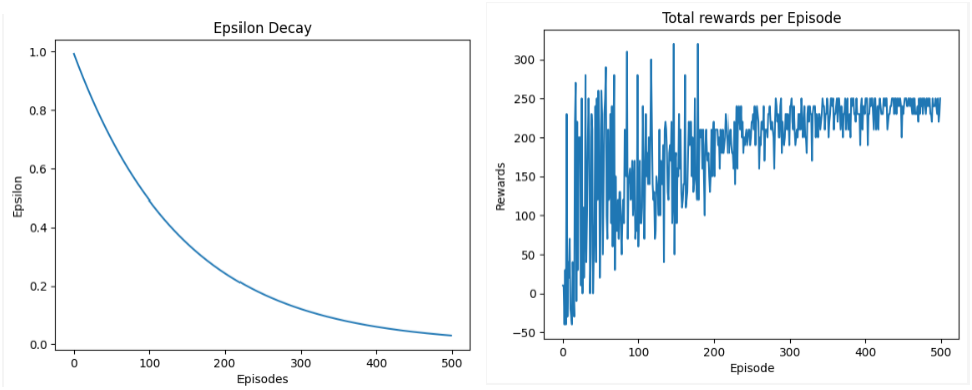
The Epsilon decay graph shows the gradual reduction of the exploration parameter (epsilon) over time in SARSA.

The total rewards are varies in the beginning then becomes constant which shows that our agent is learning the environment.

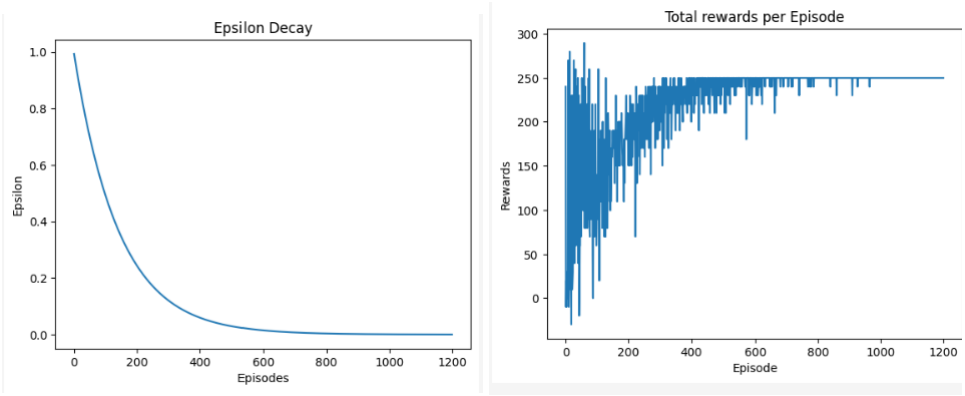
Hyperparameter tuning.

1. Changing the number of episodes.

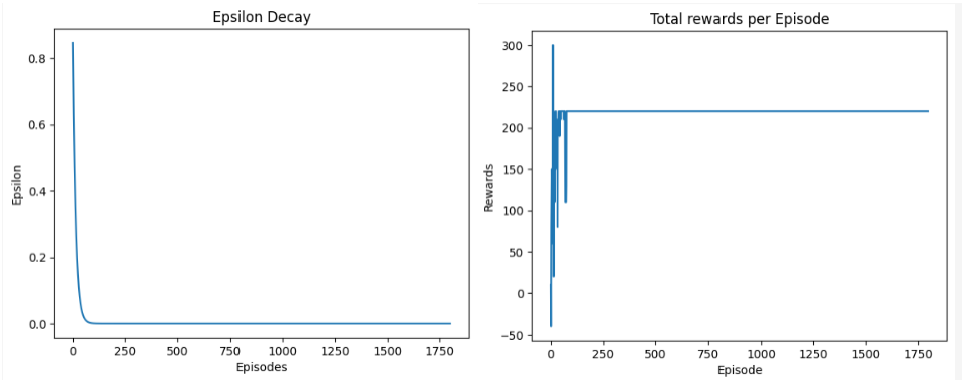
a. For 500 episodes



b. For 1200 Episodes

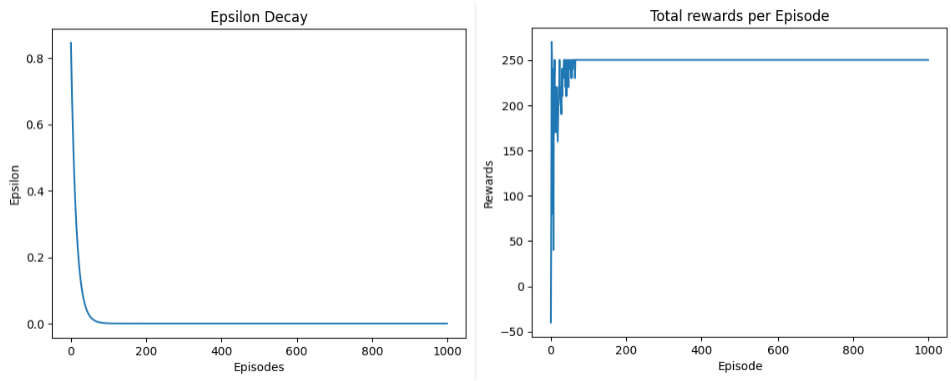


c. For 1800 Episodes

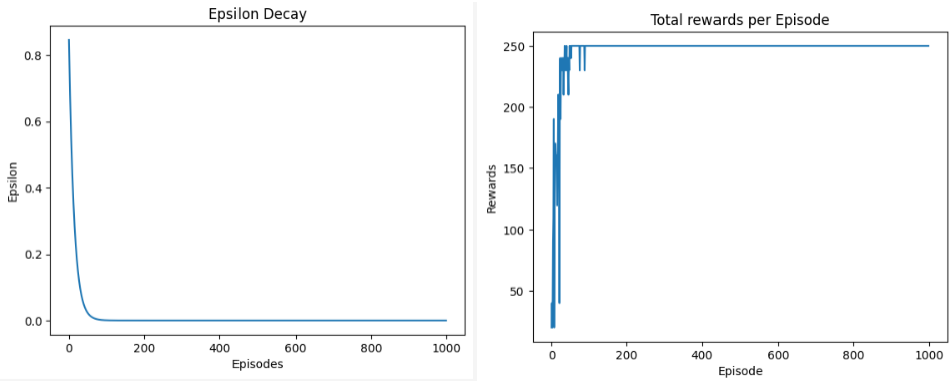


1. Changing the discount factor.

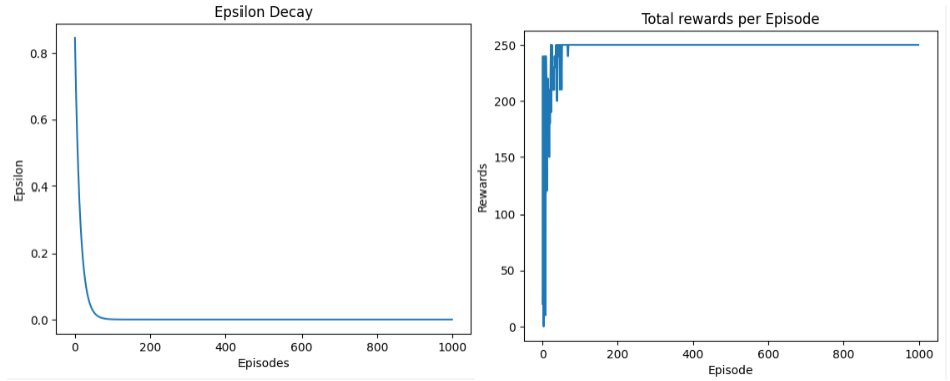
a. For discount Factor 0.2



b. For discount Factor 0.5



c. For discount Factor 0.75



Q-Learning

```python
import numpy as np
import random

class QLearningAgent:

    def __init__(self, grid, alpha=0.1, gamma=0.99, epsilon=0.98, epsilon_decay=0.009, episodes=1000):
        self.dict_q1 = {}
        self.alpha = alpha
        self.gamma = gamma
        self.epsilon = epsilon
        self.epsilon_decay = epsilon_decay
        self.episodes = episodes
        self.stateMap = grid
        self.arr = []
        for state in grid.values():
            self.dict_q1[state] = np.zeros(4)

    def choose_nextaction(self, q_values, epsilon=0.01):
        actions = [0, 1, 2, 3]
        if np.random.rand() < epsilon:
            selected_action = random.choice(actions)
        else:
            selected_action = np.argmax(q_values)
        return selected_action

    def Q_value_calculation(self, reward, new_state, dict_q1, action, old_state):
        new_state = grid.get(tuple(new_state["position"]))
        q_old = dict_q1[old_state][int(action)]
        max_q_new_action = np.max(dict_q1[new_state])
        dict_q1[old_state][int(action)] += self.alpha * (reward + (self.gamma * max_q_new_action) - q_old)
        return dict_q1[old_state][int(action)]
```

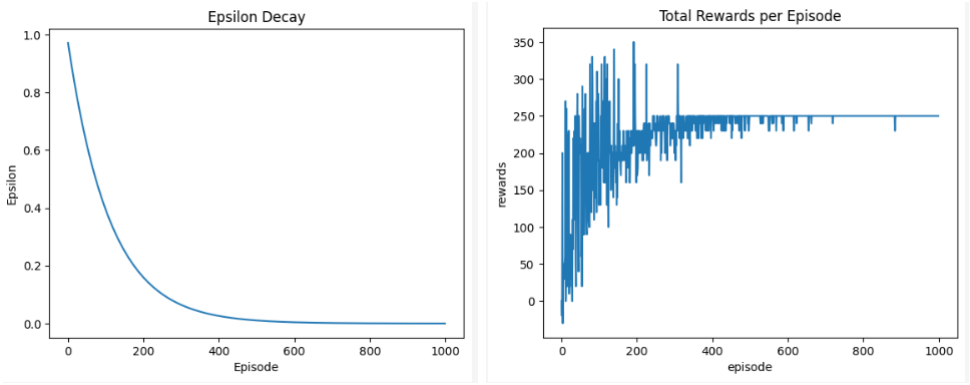
```python
grid = {
    (0, 0): 0, (0, 1): 1, (0, 2): 2, (0, 3): 3,
    (1, 0): 4, (1, 1): 5, (1, 2): 6, (1, 3): 7,
    (2, 0): 8, (2, 1): 9, (2, 2): 10, (2, 3): 11,
    (3, 0): 12, (3, 1): 13, (3, 2): 14, (3, 3): 15
}

qlearning_Agent=QLearningAgent(grid=grid, alpha=0.1, gamma=0.99, epsilon=0.98, epsilon_decay=0.009, episodes=1000)

arr, final_res = qlearning_Agent.train()

plt.plot(arr)
plt.title('Epsilon Decay')
plt.ylabel('Epsilon')
plt.xlabel('Episode')
plt.show()

plt.plot(final_res)
plt.title('Total Rewards per Episode')
plt.ylabel('rewards')
plt.xlabel('episode')
plt.show()
```



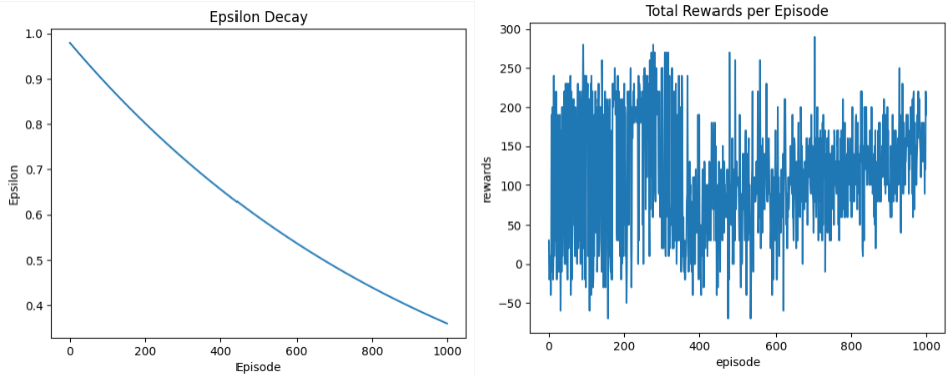
The Epsilon decay graph shows the gradual reduction of the exploration parameter (epsilon) over time in Q-Learning as well.

The total rewards varies in the beginning then becomes constant which shows that our agent is learning the environment.

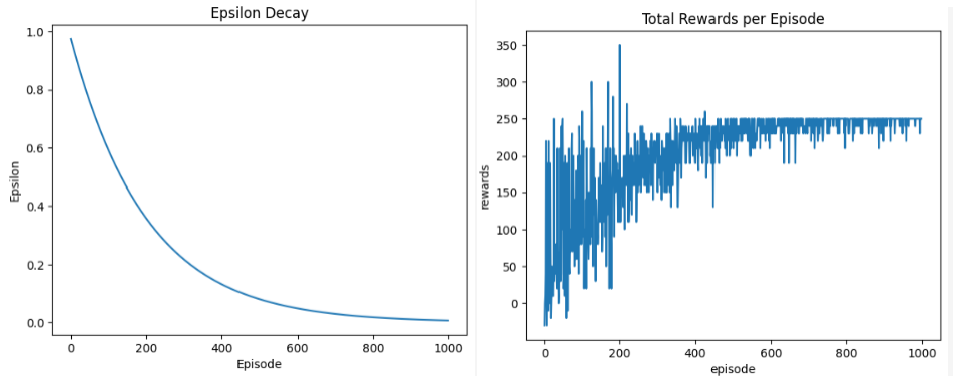
Hyperparameter tuning.

1. Changing the epsilon decay.

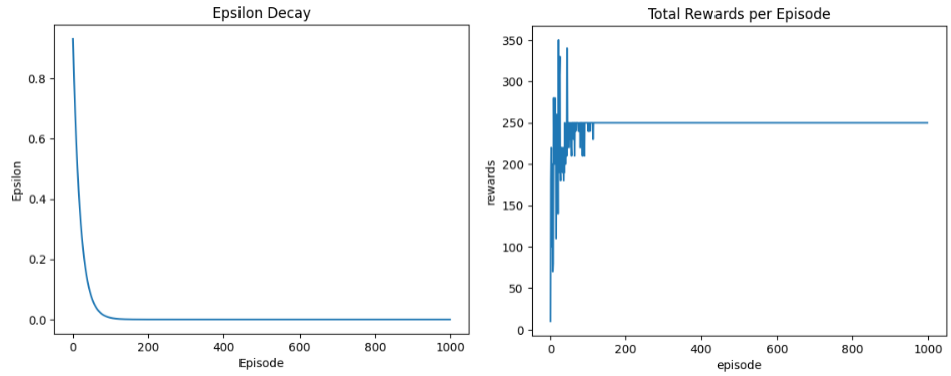
a. For decay at 0.001



b. For decay at 0.005

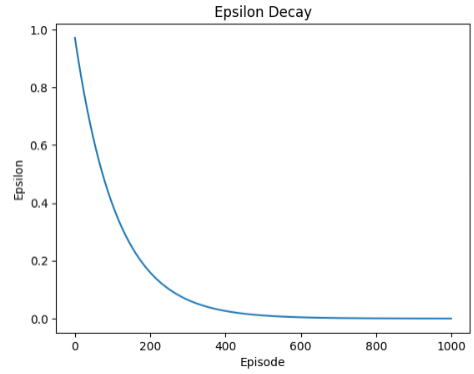


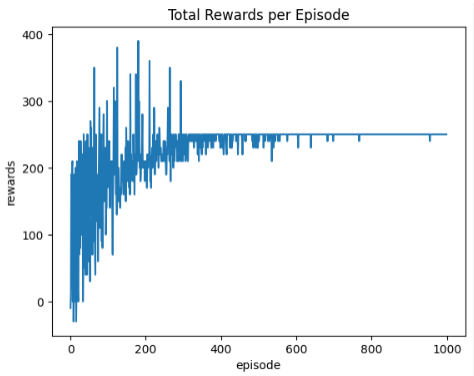
c. For decay at 0.05



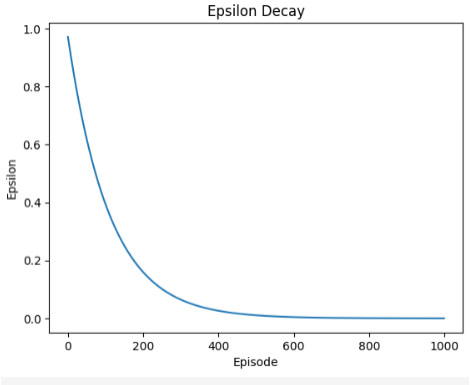
2. Changing the discount factor.

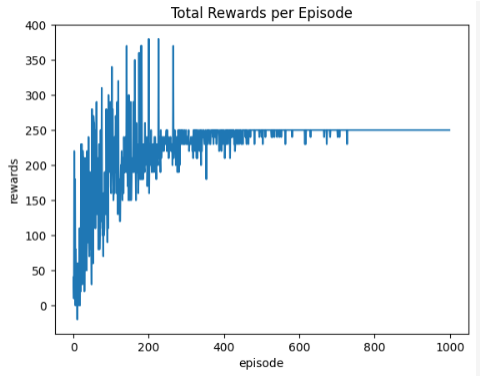
a. Discount factor at 0.5



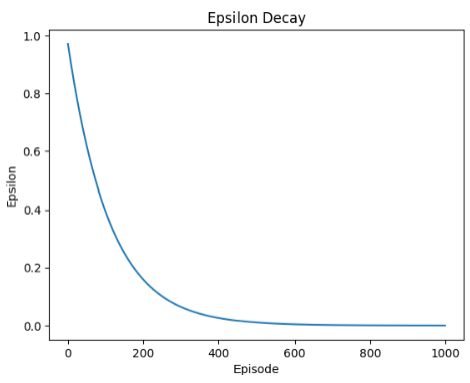


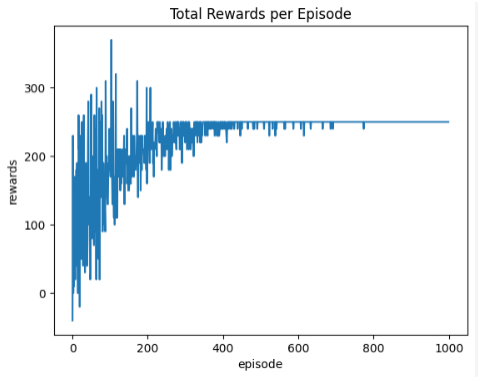
b. Discount factor at 0.1





c. Discount factor at 0.7



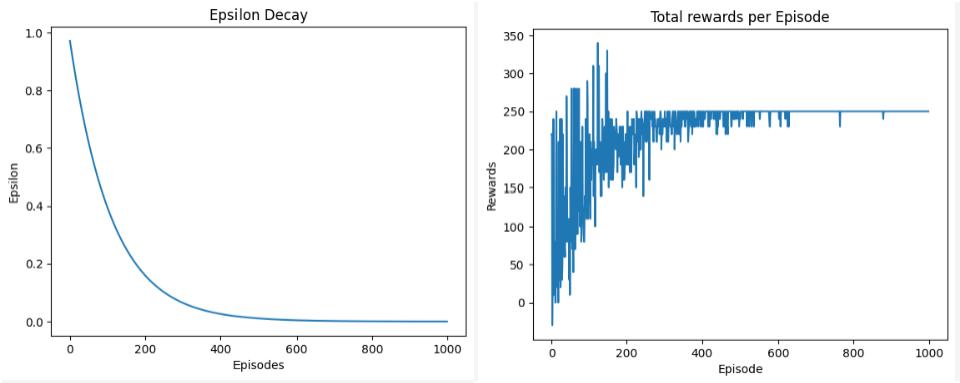


The above graph shows similar plotting unless the hyperparameter is changed to the extreme.

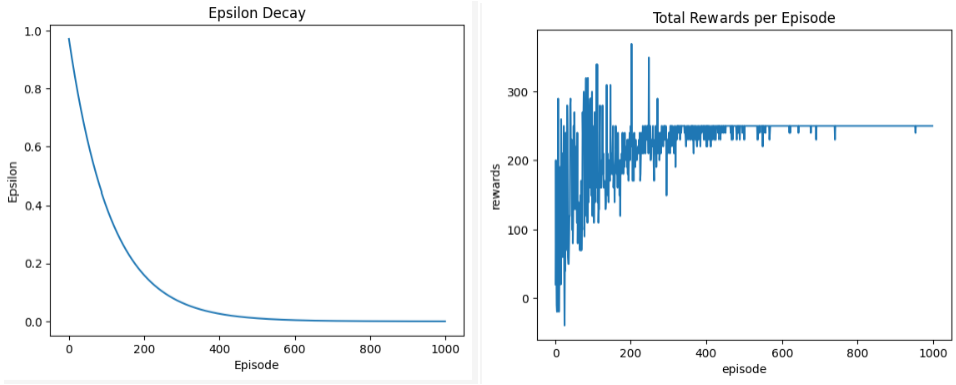
Only greedy steps

The next step function is changed to take only the best step from the q stable, so following are the plots we get.

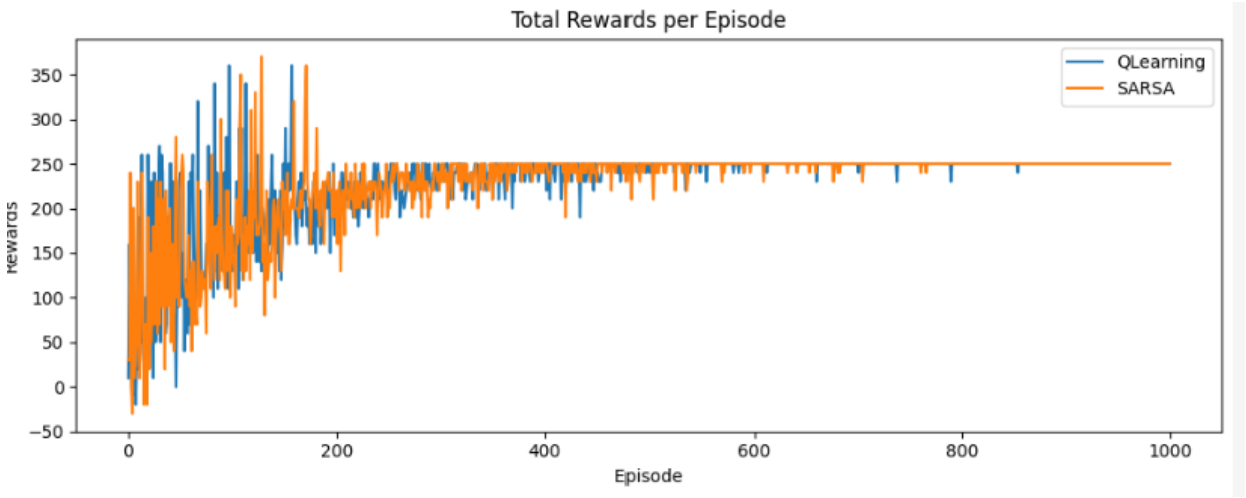
SARSA



Q-Learning



Plotting together the reward dynamics



The above graph shows that Q learning has more variation initially as compared to SARSA after that both the algorithm converges.

The Best parameters for SARSA

```
sarsa_agent=SarsaAgent(grid=grid, alpha=0.3, gamma=0.75, epsilon=0.91, epsilon_decay=0.07, episodes=1000)
```

The Best parameters for Q-Learning

```
qlearning_Agent=QLearningAgent(grid=grid, alpha=0.1, gamma=0.99, epsilon=0.98, epsilon_decay=0.05, episodes=1000)
```

REFERENCES

WEBSITES

- https://medium.com/@harshitsikchi/towards-safe-reinforcement-learning-88b7caa5702e
- https://aitechtrend.com/all-you-need-to-know-about-sarsa-in-reinforcement-learning/#:~:text=SARSA%20is%20a%20powerful%20algorithm%20in%20Reinforcement%20Learning%20that%20has,converging%20to%20an%20optimal%20policy.
- https://ubuffalo-my.sharepoint.com/personal/avereshc_buffalo_edu/_layouts/15/onedrive.aspx?id=%2Fpersonal%2Favereshc%5Fbuffalo%5Fedu%2FDocuments%2F2023%5FSpring%5FML%2F%5Fpublic%2FCourse%20Materials%2FCoding%20Tutorials%2FRL%20Environment%20Visualization%20by%20Nitin%20Kulkarni&ga=1