**ML Assignment 3**

**Report Part 2 & 3**

1. **Briefly explain the tabular methods that were used to solve the problems. Provide their update functions and key features. What are the advantages/disadvantages?**

For SARSA:

* The SARSA algorithm is a tabular reinforcement learning method that was used to solve the problem in the provided code. Using the following update rule, it modifies the Q-values for state-action pairs based on the current reward and the Q-value of the subsequent state-action pair:

Q(s,a) = Q(s,a) \* (reward + gamma) + alpha Q(s',a') = Q(s,a)

where Q(s,a) denotes the Q-value for a certain state-action pair (s,a), alpha denotes the learning rate, reward is the reward earned after acting in state s, gamma denotes the discount factor, and Q(s',a') denotes the Q-value for the subsequent state-action combination.

Here, SARSA was used to learn the optimal policy for a simple gridworld problem. The Q-table was initialized to -1, and epsilon-greedy exploration was used to balance between exploration and exploitation. During training, the agent chooses actions based on the epsilon-greedy policy, updates the Q-values for the current state-action pair using the SARSA update rule, and moves to the next state. This process is repeated until the agent reaches the terminal state. The agent's performance is then evaluated during testing, where it chooses actions based on the greedy policy (i.e., the action with the highest Q-value for a given state).

Advantage:

One advantage of tabular methods like SARSA is that they are simple to implement and easy to interpret, as they rely on a table to store Q-values for each state-action pair. Additionally, they are model-free, meaning they can learn directly from experience without requiring a model of the environment.

Disadvantage:

Tabular methods may not scale well to larger problems with high-dimensional state and action spaces, and may require a large number of training episodes to converge to an optimal policy.

For Q-learning:

Key Features:

Q-table: The Q-learning algorithm uses a Q-table to store the Q-values for each state-action pair.

Epsilon-Greedy Exploration: The code implements epsilon-greedy action selection, which balances exploration and exploitation by choosing a random action with probability epsilon and choosing the action with the highest Q-value with probability (1 - epsilon).

Q-value Update: The Q-values are updated using the Q-learning update rule, which incorporates the immediate reward and the maximum Q-value of the next state.

Training and Testing: The code includes methods for training the Q-learning model over a specified number of episodes and testing the trained model.

Benefits of Q-learning

* Model-Free: Since Q-learning is a model-free reinforcement learning algorithm, it does not need to be explicitly aware of the dynamics of the environment. It learns immediately via encounters with the environment.
* Convergence: Q-learning is guaranteed to converge to the optimal policy for finite Markov Decision Processes (MDPs) under certain conditions.
* Off-Policy Learning: Q-learning is an off-policy learning algorithm, which enables it to draw conclusions from data produced by any policy, even one that is not being updated.

Disadvantages:

* Large State Spaces: Since the Q-table must be able to hold the Q-values for each state-action pair, problems with large state spaces make Q-learning computationally and memory-intensive.
* Exploration-Exploitation Tradeoff: It might be difficult to strike the correct balance between exploration and exploitation. The exploration rate is controlled by the epsilon parameter, and choosing the right value is essential to ensuring that there is enough exploration without being trapped in ineffective policies.
* Continuous State and Action Spaces: Q-learning cannot be used to solve continuous state and action space issues directly. The state and action spaces must be discretized, which could result in data loss and insufficient performance.

Q-learning Update Function:

The train method in the code contains the Q-learning Update Function. The temporal difference (TD) error, which is the difference between the updated Q-value based on the immediate reward and the maximum Q-value of the following state, is calculated by this update function. It then uses the learning rate (alpha) and TD error to update the Q-value for the current state-action combination.

Overall, the code offers a simple Q-learning implementation that may be used as a jumping off point for testing out reinforcement learning problems. To handle more complicated settings and boost performance, it might need additional tweaks and improvements.

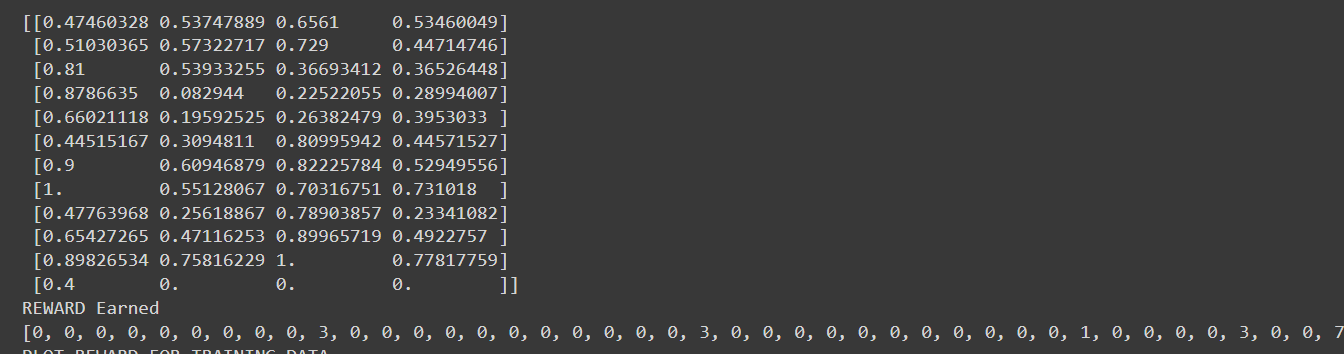
**2. Show and discuss the results after:**

**• Applying SARSA to solve the environment defined in Part 1. Plots should include epsilon decay and total reward per episode.**

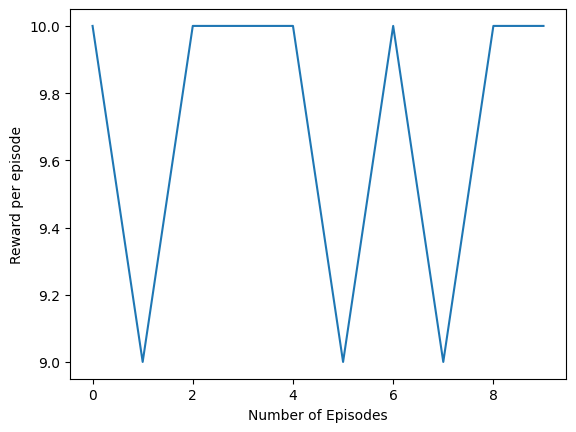
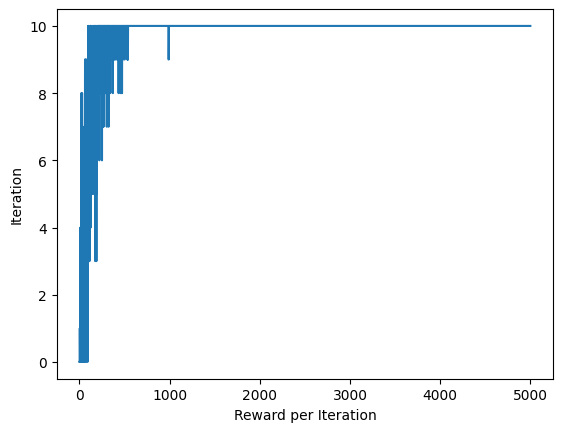
**• Applying Q-learning to solve the stochastic environment defined in Part 1. Plots should include epsilon decay and total reward per episode.**

**• Provide the evaluation results. Run your environment for at least 10 episodes, where the agent chooses only greedy actions from the learned policy. Plot should include the total reward per episode.**

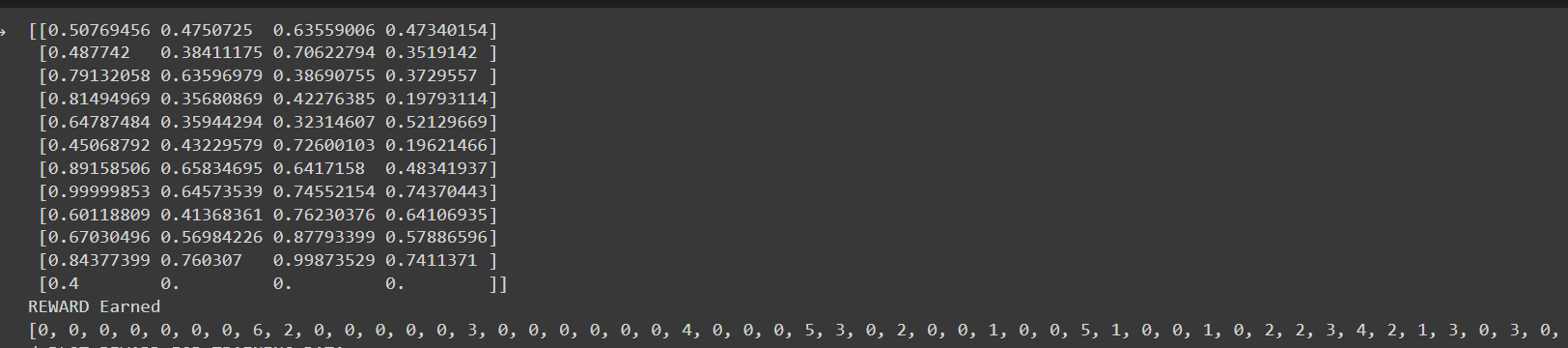
For SARSA:

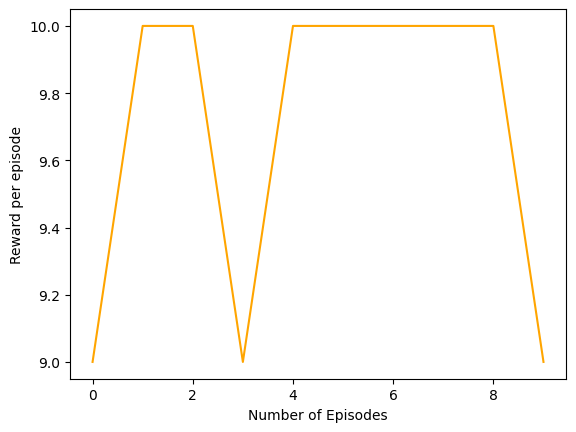
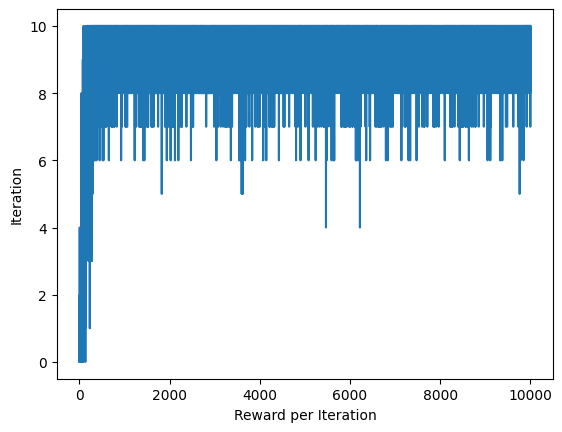


The following plots are for 5000 episodes, gamma = 0.9, epsilon = 1, alpha = 0.4

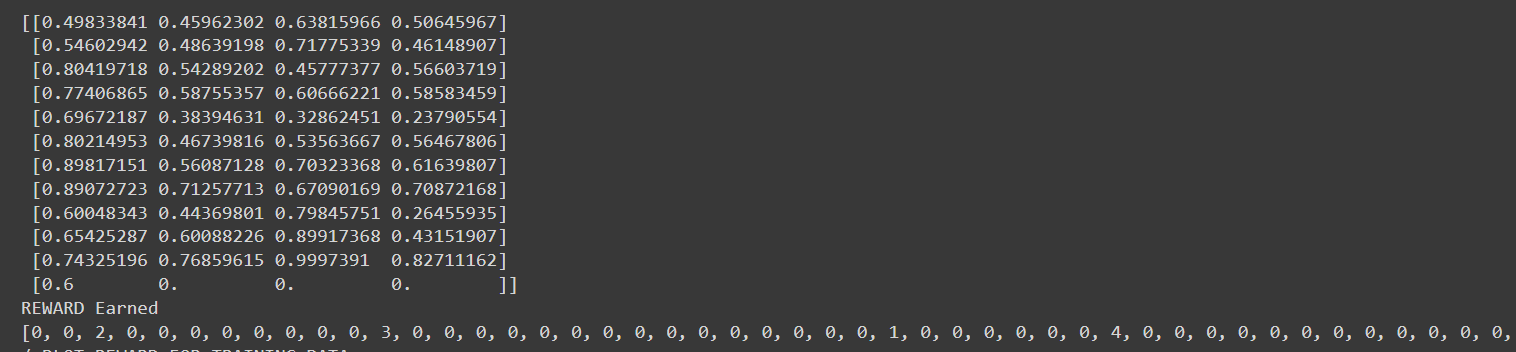


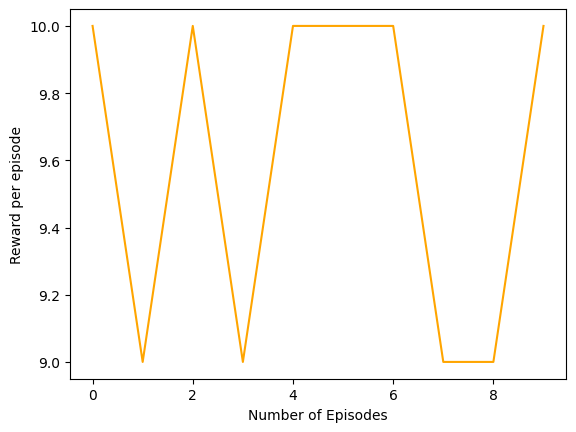
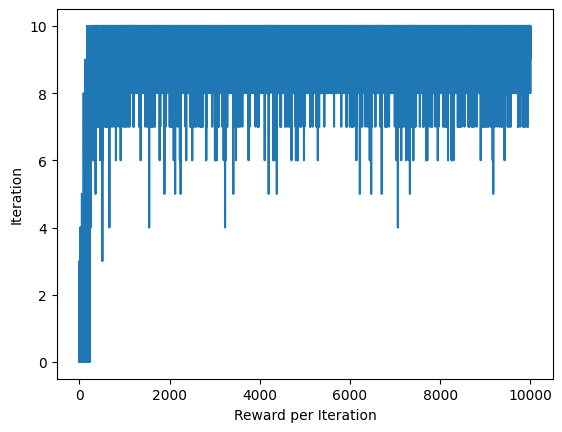
The following plots are for 10000 episodes, gamma = 0.9, epsilon = 1, alpha = 0.4 but we use Stochastic movement





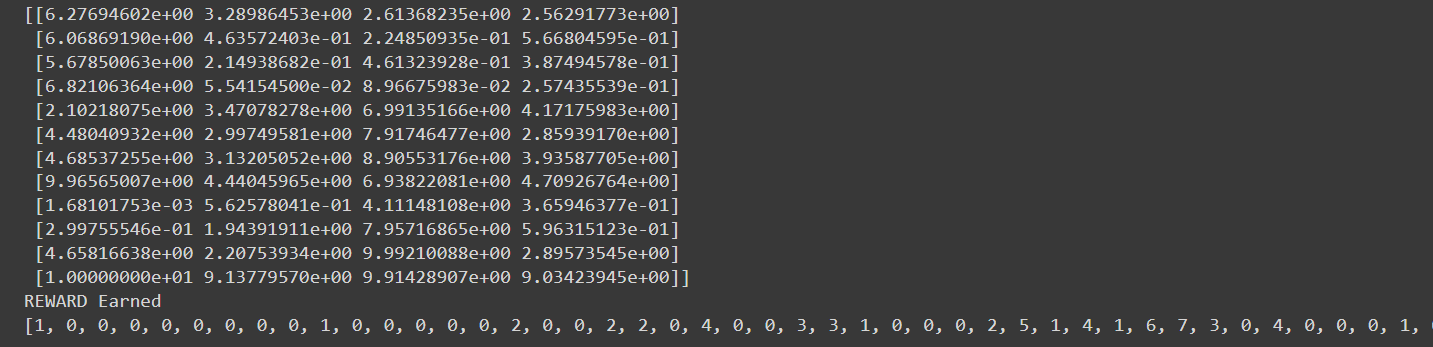
The following plots are for 10000 episodes, gamma = 0.9, epsilon = 2, alpha = 0.6 but we use Stochastic movement

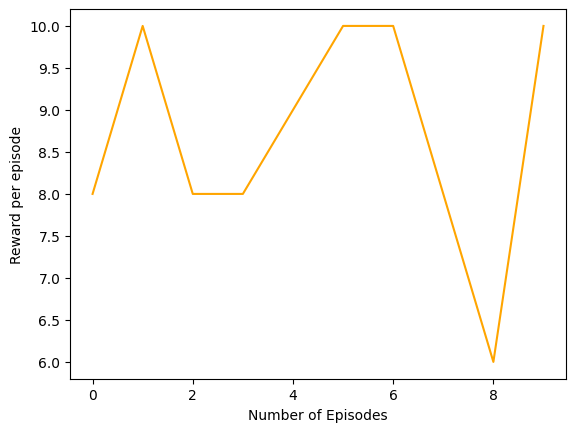
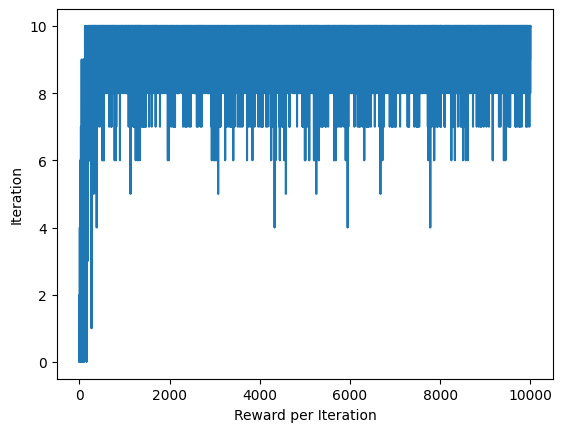




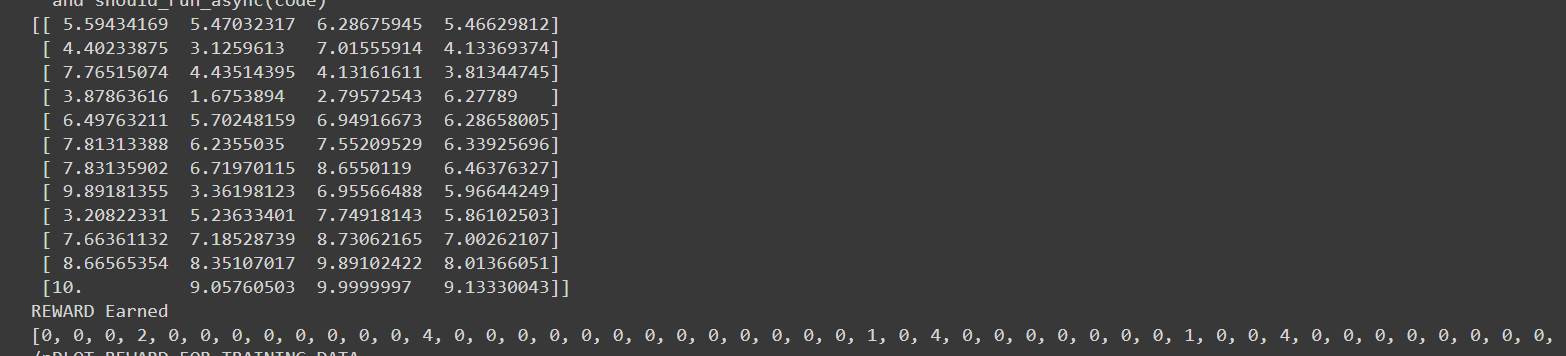
**For Q-Learning**:

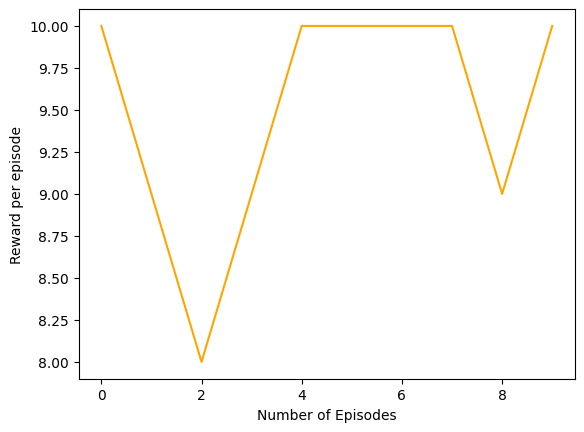
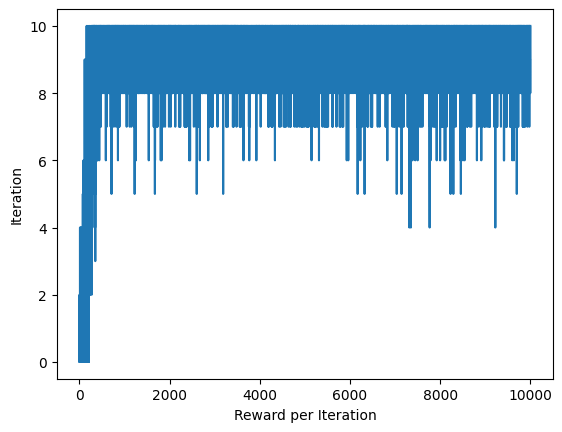
The following plots are for 10000 episodes, gamma = 0.9, epsilon = 1, alpha = 0.1 but we use Deterministic movement



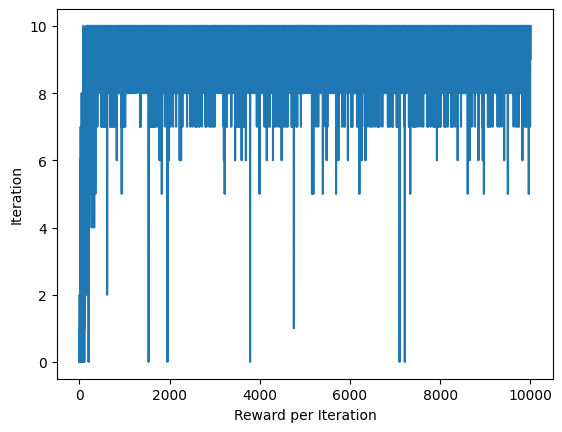
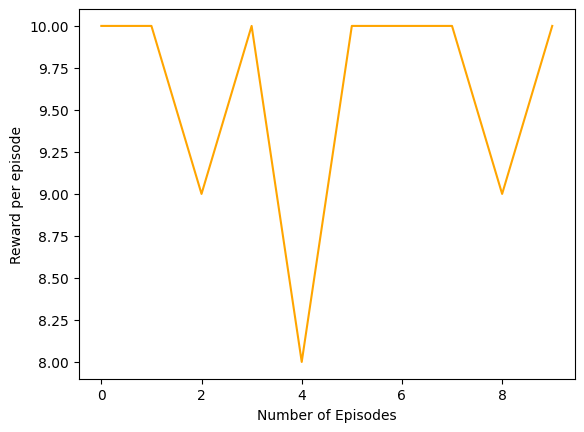


The following plots are for 10000 episodes, gamma = 0.9, epsilon = 2, alpha = 0.3 but we use Stochastic movement





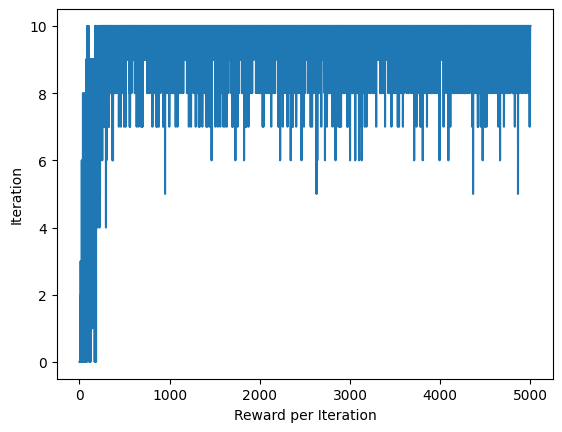
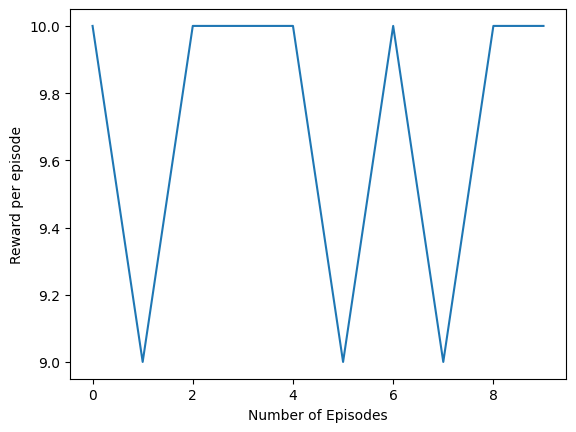
The following plots are for 10000 episodes, gamma = 0.9, epsilon = 1, alpha = 0.8 but we use Stochastic movement

**3. Compare the performance of both algorithms on the same environment (e.g. show one graph with two reward dynamics) and give your interpretation of the results.**

For SARSA:

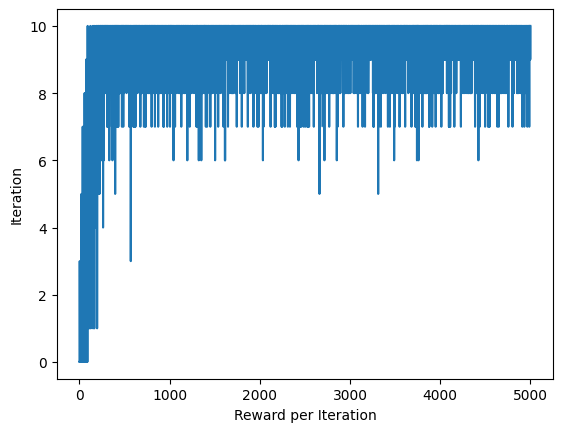
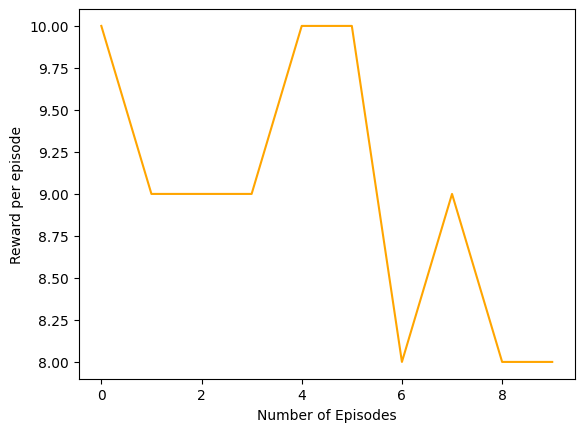
The following plots are for 5000 episodes, gamma = 0.9, epsilon = 1, alpha = 0.1 but we use Stochastic movement:

SARSA algorithm took 5000 episodes and has a range of 1-9 timesteps per episode. The reward is always stable at 10. So, the highest reward is 10 per episode

For Q-learning:

The following plots are for 5000 episodes, gamma = 0.9, epsilon = 1, alpha = 0.1 but we use Stochastic movement:

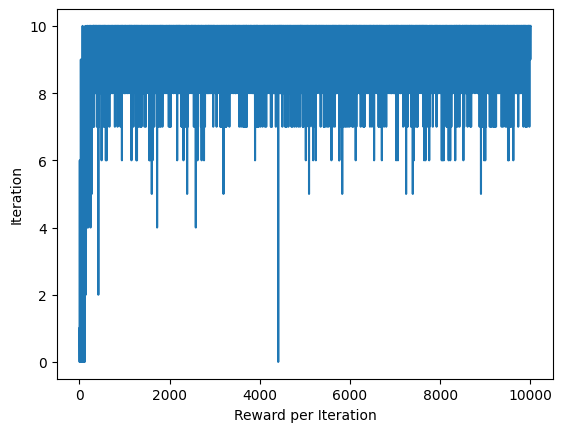
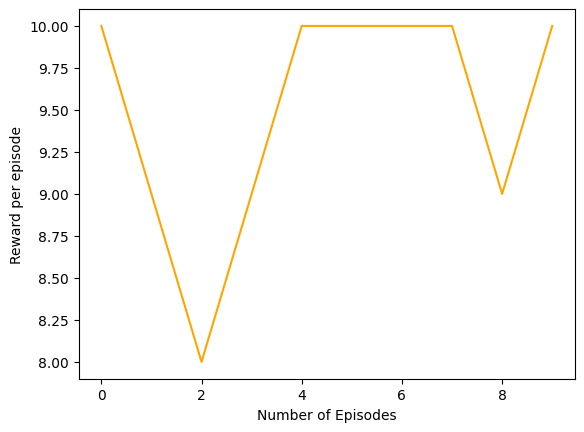
 

Q-learning algorithm took 5000 episodes and has a range of 1-9 timesteps per episode.

The reward is always stable at 7. So, the highest reward is 7 per episode.

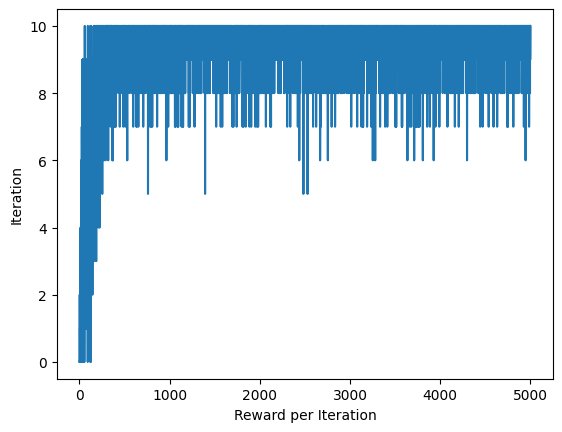
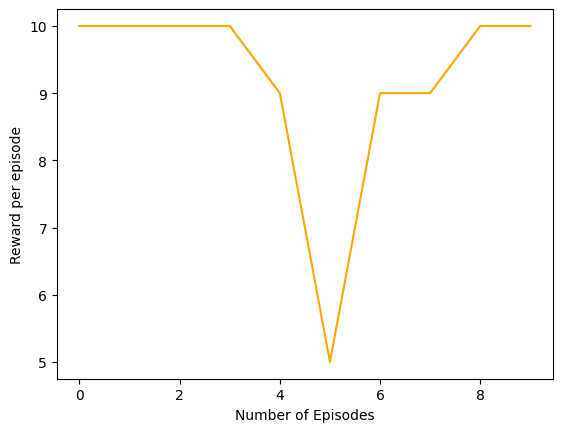
For SARSA:

The following plots are for 10000 episodes, gamma = 0.9, epsilon = 1, alpha = 0.4 but we use Stochastic movement

SARSA algorithm took 10000 episodes and has a range of 5-9 timesteps per episode. Here, we see that the highest reward is 10 per episode.

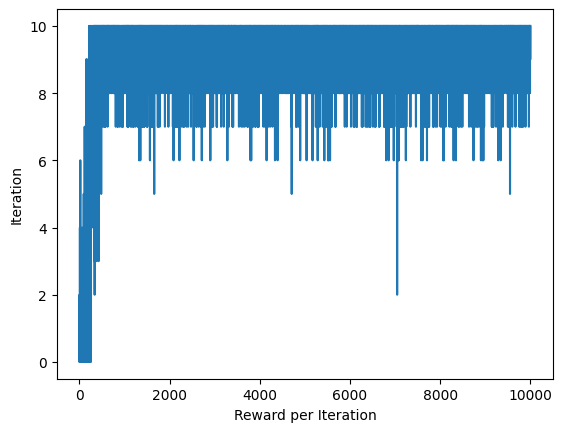
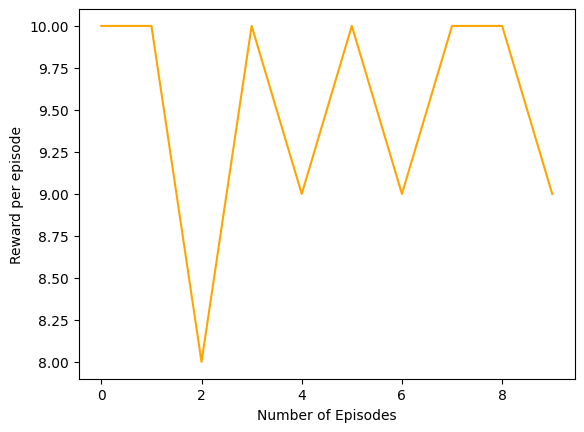
For Q-learning: The following plots are for 10000 episodes, gamma = 0.9, epsilon = 1, alpha = 0.4 but we use Stochastic movement

Q learning algorithm took 10000 episodes and has a range of 5-9 timesteps per episode. Here, we see that the highest reward is 10 per episode.

For SARSA:

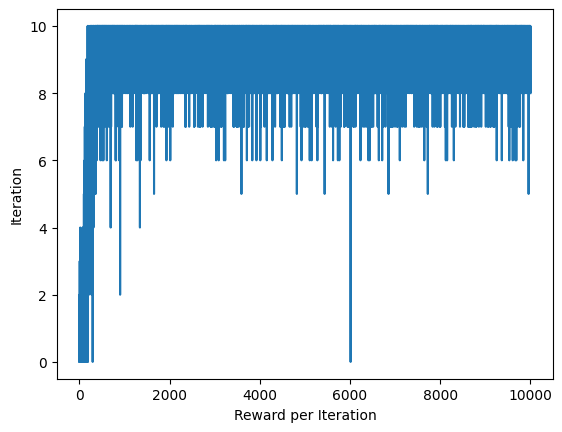
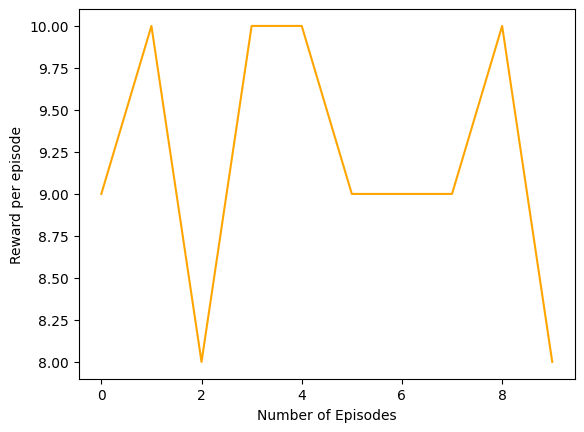
The following plots are for 10000 episodes, gamma = 0.9, epsilon = 2, alpha = 0.6 but we use Stochastic movement

The SARSA algorithm takes 10000 episodes and has a range of 4 – 8 timesteps per episode. We can see that the highest reward is 10 per episode.

For Q-learning:

The following plots are for 10000 episodes, gamma = 0.9, epsilon = 2, alpha = 0.6 but we use Stochastic movement

The Q-learning algorithm takes 10000 episodes and has a range of 4 – 8 timesteps per episode. We can see that the highest reward is 9 per episode.

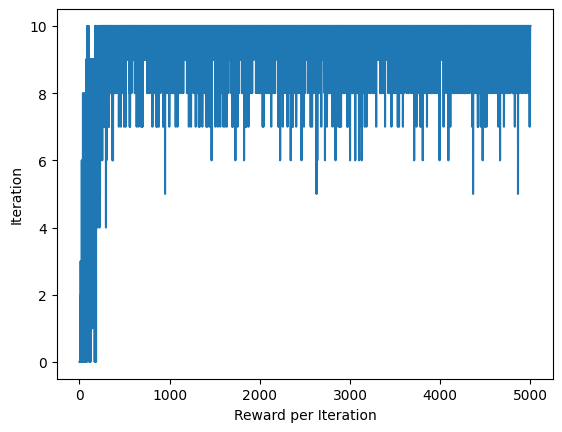
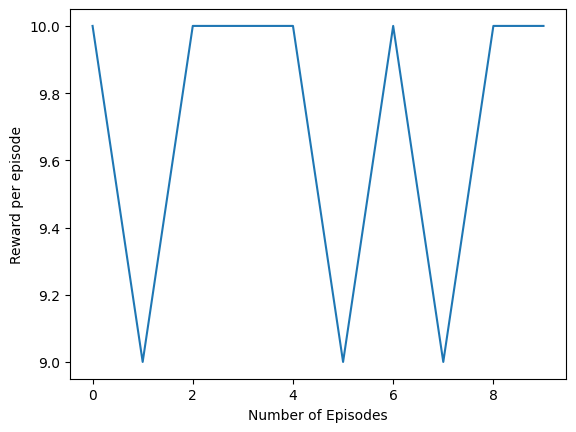
From all this we can see a consistent reward score in SARSA model compared to Q-learning.

**4. Provide the analysis after tuning at least two hyperparameters from the list above. Provide the reward graphs and your explanation for each of the results. In total, you should have at least 6 graphs for each implemented algorithm and your explanations. Make your suggestion on the most efficient hyperparameters values for your problem setup.**

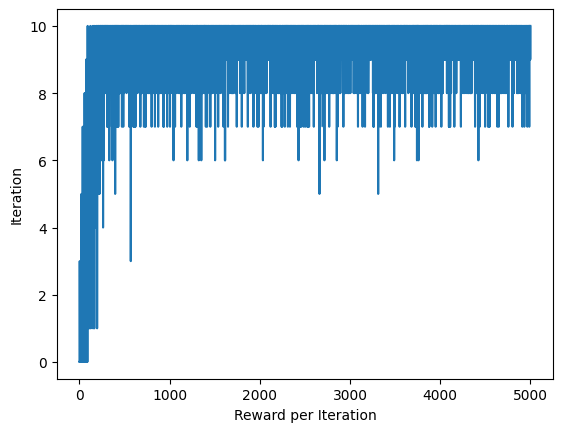
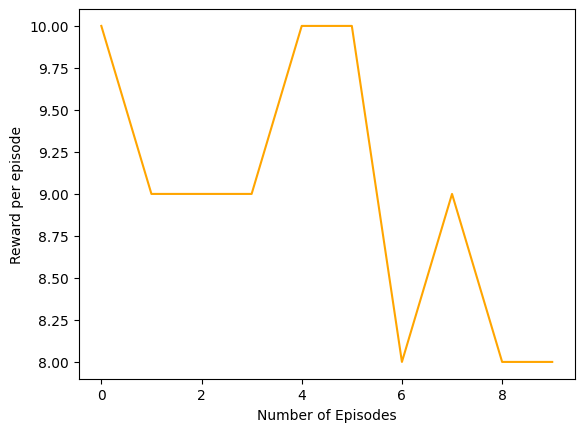
Epsilon Value: 1

Alpha Value: 0.1

SARSA:

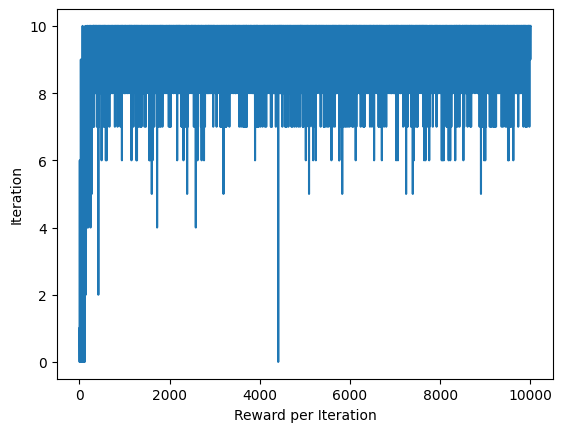
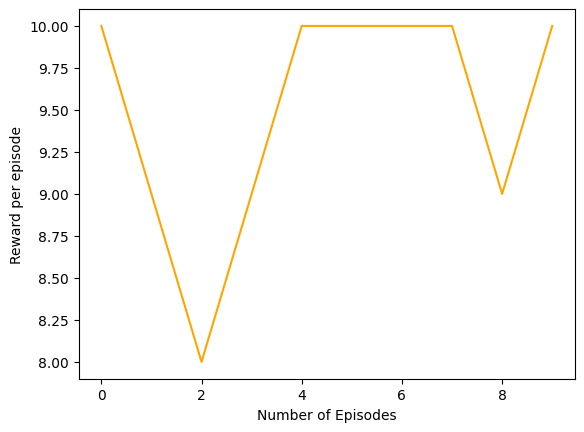
Q-learning:

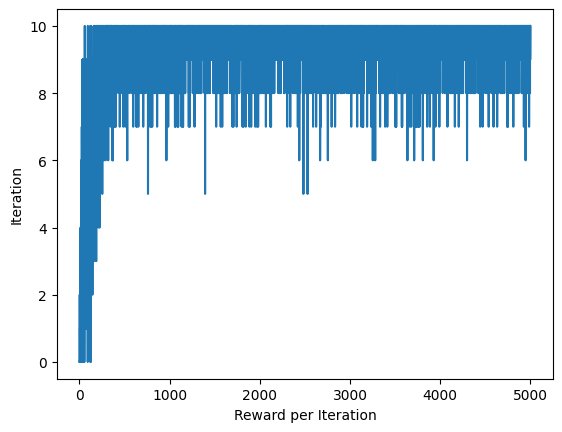
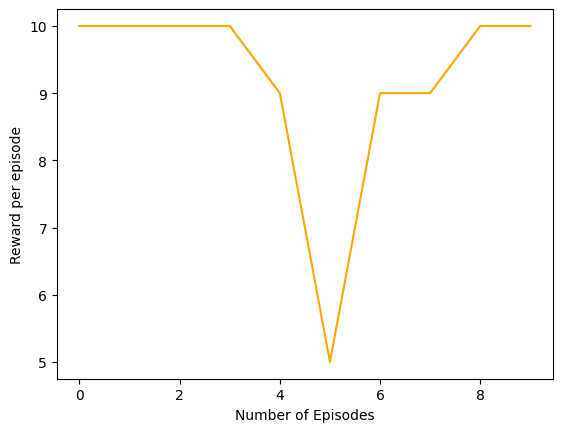
Epsilon Value: 1

Alpha Value: 0.4

SARSA:

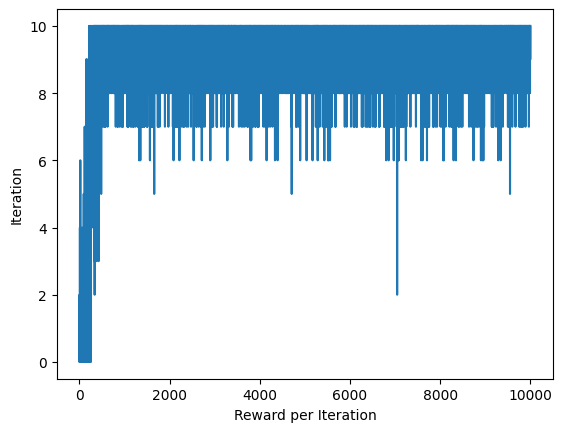
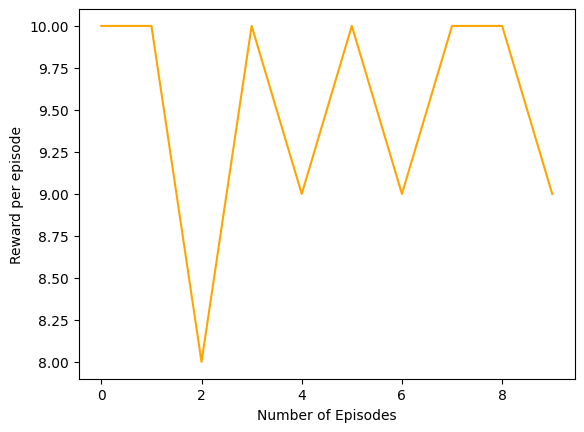
Q-learning:

Epsilon Value: 2

Alpha Value: 0.6

SARSA:

Q-learning

