# CS6700
# Reinforcement Learning (Spring 2024)
# Programming Assignment 1

SARSA and *Q*-Learning

*Instructor: Prof. Balaraman Ravindran*

*Students:*
*ME20B087 Janmenjaya Panda*
*ME20B122 Nishant Sahoo*

**Release Date: 14/02/2024**

**Submission Deadline: 28/03/2024**


**Department of Computer Science & Engineering**
**Indian Institute of Technology Madras**

# Contents

# 1 Environment Description

This exercise aims to study the two popular Temporal Difference Learning algorithms: **SARSA**[1] and ***Q*-Learning**. We will solve several variants of the Grid World problem (a sample world is shown in Figure 1).

This is a grid world with 4 deterministic actions ('up', 'down', 'left', 'right'). The agent transitions to the next state determined by the direction of the action chosen with a probability of $p \in [0, 1]$. We also define a parameter called $b \in [0, 1]$. Consider the direction of the action chosen as the agent's "North". For example, if the action is 'left', it is the agent's North, and the agent's East would be the direction of the action 'up'. Figure 1 provides an illustration of the same. The agent transitions to the state West of the chosen action with probability $(1 - p) \times b$, and to the East of the chosen action with probability $(1 - p) \times (1 - b)$.

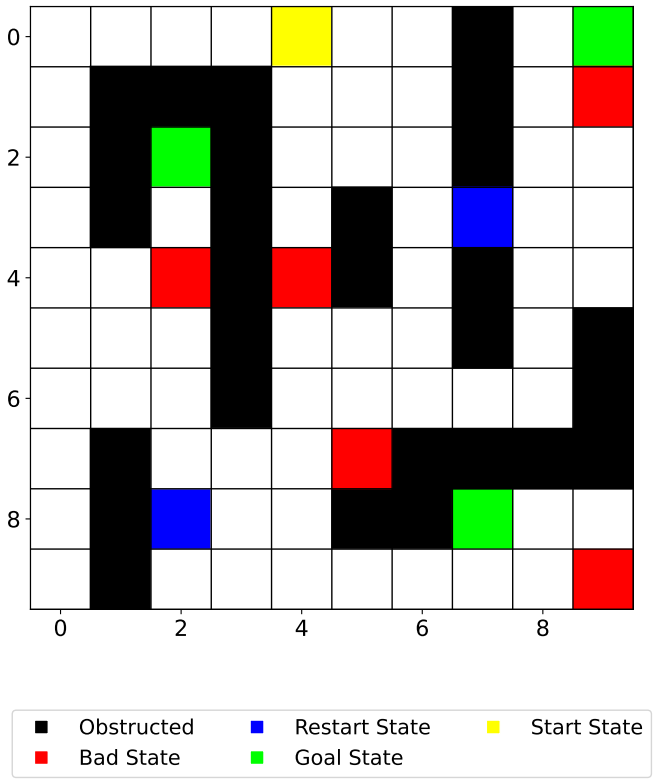



Figure 1: An example grid with start point at (0, 4)

The environment may also have a wind blowing that can push the agent one **additional** cell to the right **after transitioning to the new state** with a probability of 0.4. An episode is terminated either when a goal is reached or when the timesteps exceed 100. Transitions that take you off the grid will not result in any change in state. The dimensions of the grid are $10 \times 10$. The following types of states exist:

- ***Start state:*** The agent starts from this state.
- ***Goal state:*** The goal is to reach one of these states. There are 3 goal states in total.
- ***Obstructed state:*** These are walls that prevent entry to the respective cells. Transition to these states will not result in any change.

[1] State - Action - Reward - State - Action

- ***Bad state:*** Entry into these states will incur a higher penalty than a normal state.
- ***Restart state:*** Entry into these states will incur a very high penalty and will cause the agent to teleport to the start state without the episode ending. Once the restart state is reached, no matter what action is chosen, it goes to the start state at the next step.
- ***Normal state:*** None of the above. Entry into these states will incur a small penalty.

Rewards: -1 for normal states, -100 for restart states, -6 for bad states, +10 for goal states.

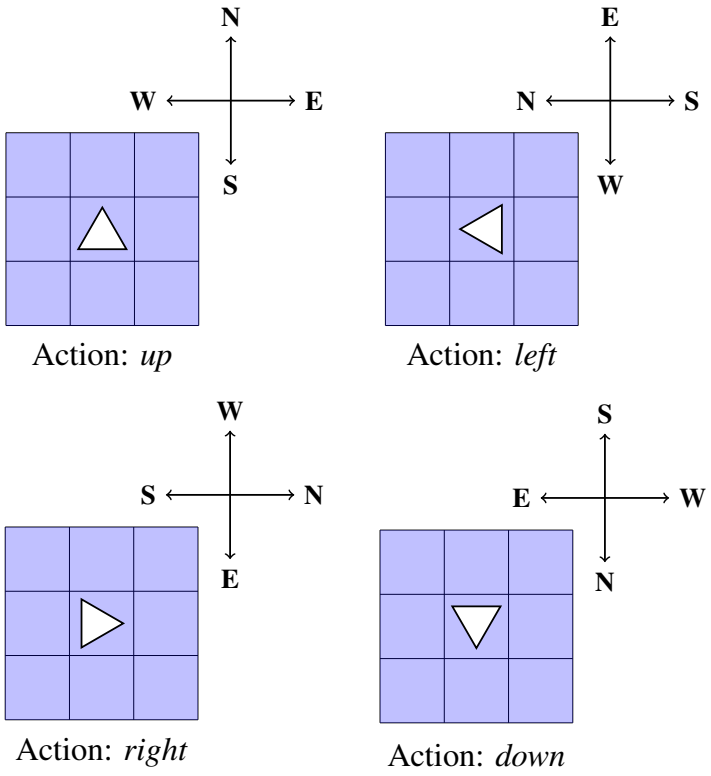



Table 1: The intended direction of the action chosen is considered as 'North'

# 2 Additional Information

The code for the environment can be found here. The `env.step()` function takes as arguments the current state and action and returns the reward and next state. The appropriate termination conditions have to be specified by the student in the code (as explained in section 1, `env.reset()` resets the environment. In the notebook containing the code, cell 1 contains the environment class, cell 2 contains the environment instantiation, and cell 3 lists some environment variables. For each experiment, the start state is fixed and does not change. Different experiments may have different start states.

# 3 Problem 1

Implement SARSA and $Q-$Learning algorithms.

---

## 3.1 SARSA

### 3.1.1 What is SARSA?

SARSA stands for State−Action−Reward−State−Action. It is an on-policy[2] designed for solving MDPs[3]. It learns a policy by interacting with an environment and updating $Q-$values based on observed rewards and actions taken.

### 3.1.2 Why is SARSA Used?

SARSA is used for:

- On-policy learning where the policy used for exploration matches the policy used for learning.
- Balancing exploration and exploitation through an epsilon-greedy strategy.
- Adapting to changing environments by continuously updating $Q-$values.

### 3.1.3 The Algorithm

**Algorithm 1** : SARSA Algorithm (Source: [1])

Initialize $Q(s, a)$ arbitrarily
**for** each episode **do**
    Initialize $S$
    Choose action $A$ from state $S$ using policy ($\epsilon-$greedy/ softmax) derived from $Q$
    **while** $S$ is not the goal state and max steps not reached **do**
        Take action $A$, observe reward $R$ and next state $S'$
        Choose next action $A'$ from $S'$ using policy derived from $Q$
        $\boxed{Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma Q(S',A') - Q(S,A)\right]}$
        $S \leftarrow S'; A \leftarrow A'$
    **end while**
**end for**

### 3.1.4 An overview

**Initialization Phase**:

- $Q-$**Value Initialization**: Initialize $Q-$values for all state-action pairs arbitrarily.

**Exploratory Phase**:

[2] policy used for exploration is the same as the policy used for learning [3] Markov Decision Processes

- **Action Selection**: In each state, the agent selects an action using an epsilon-greedy policy for exploration and exploitation.
- **Action Execution and Environment Interaction**: The agent executes the chosen action, and the environment responds with a reward and transitions to a new state.

**Learning Phase**:

- **Next Action Selection**: Based on the new state, the agent selects the next action using the epsilon-greedy policy.
- $Q-$**Value Update**: Update the $Q-$value of the current state-action pair using the SARSA update rule:

$$Q(S, A) \leftarrow Q(S, A) + \alpha \left[R + \gamma Q(S', A') - Q(S, A)\right]$$

**Repeat and Termination**:

- **Repeat Steps 2 and 3**: Continue exploration, action selection, and learning until reaching a terminal state or predefined iterations.
- **Termination**: The algorithm terminates when reaching a terminal state or after a predefined number of iterations.

### 3.1.5 The idea Behind SARSA

The fundamental idea is to estimate the expected cumulative reward of following a policy in a given environment. SARSA achieves this by iteratively updating $Q-$values based on observed rewards and the $Q-$values of subsequent state-action pairs.

### 3.1.6 Pros of SARSA

1. **On-Policy Stability**: SARSA provides stability in learning due to its on-policy nature.
2. **Adaptability to Changing Environments**: SARSA can adapt to changes by continuously updating $Q-$values.
3. **Balance of Exploration and Exploitation**: Epsilon-greedy strategy helps in balancing exploration and exploitation.

### 3.1.7 Cons of SARSA

1. **Exploration Challenges**: SARSA may struggle in environments requiring extensive exploration.
2. **Sample Inefficiency**: It might require a large number of samples to converge.
3. **Limited to Finite MDPs**: SARSA is designed for finite MDPs and may face challenges in continuous or large state and action spaces.

### 3.1.8 Implementation of SARSA

Under the assumptions of the standard representation for Python libraries[4], the following code sets up a structure for the policy class:

```
class TDQLearner:
    def __init__(self, env, alpha, gamma, epsilon, tau, num_episodes, num_runs,
     ↪  policy='softmax'):
        """
        Constructor for TDQLearner class.

        Parameters:
        - env: The environment.
        - alpha: Learning rate.
        - gamma: Discount factor.
        - epsilon: Exploration-exploitation trade-off parameter.
        - tau: Temperature parameter for softmax policy.
        - num_episodes: Number of episodes for training.
        - num_runs: Number of runs for training.
        - policy: The policy used for action selection ('softmax' or 'epsilon_greedy').
        """
        self.env = env
        self.alpha = alpha
        self.gamma = gamma
        self.epsilon = epsilon
        self.tau = tau
        self.num_episodes = num_episodes
        self.num_runs = num_runs
        self.policy = policy

    def _choose_action(self, Q):
        """
        Choose action based on the specified policy.

        Parameters:
        - Q: $Q-$values for state-action pairs.

        Returns:
        - Selected action.
        """
        if self.policy == 'softmax':
            return softmax(Q, self.tau)  # Use softmax policy for action selection.
        elif self.policy == 'epsilon_greedy':
            return epsilon_greedy(Q, self.epsilon)  # Use epsilon-greedy policy for action
             ↪  selection.
        else:
            raise Exception("Policy not recognized")

    def train(self):
        """
        Train the TDQLearner.
        """
        pass  # Placeholder for the training algorithm (to be implemented).
```

Analogously, the following piece of Python code implements SARSA as an extension of the class

[4] Such as, `import numpy as np`

TDQLearner as explained in Algorithm 1.

```
class SARSA(TDQLearner):
    def train(self):
        # Initialize arrays to store results
        rewards = np.zeros((self.num_runs, self.num_episodes))
        steps = np.zeros((self.num_runs, self.num_episodes))
        avg_Q = np.zeros((self.env.num_states, self.env.num_actions))
        state_visits = np.zeros((self.env.num_states, 1))

        for run in range(self.num_runs):
            # Initialize $Q-$values for the current run
            Q = np.zeros((self.env.num_states, self.env.num_actions))

            for episode in tqdm(range(self.num_episodes), desc="Run: "+str(run+1), position=1):
                # Reset the environment for a new episode
                state = self.env.reset()
                done = False

                # Choose the initial action using the specified policy
                action = self._choose_action(Q[state, :])

                while not done:
                    try:
                        # Take a step in the environment and choose the next action
                        next_state, reward = self.env.step(state, action)
                        next_action = self._choose_action(Q[next_state, :])

                        # Update Q-value based on SARSA update rule
                        Q[state, action] += self.alpha * (reward + self.gamma * Q[next_state,
                        ↪  next_action] - Q[state, action])

                        # Update state and action for the next iteration
                        state = next_state
                        action = next_action

                        # Update performance metrics
                        rewards[run, episode] += reward
                        steps[run, episode] += 1
                        state_visits[state] += 1
                        # Check for termination conditions
                        if state in self.env.goal_states_seq or steps[run, episode] >
                        ↪  max_steps:
                            done = True
                    except IndexError:
                        continue # Handle index errors, if any

            # Accumulate Q-values for averaging
            avg_Q += Q

        # Average $Q-$values and state visits over all runs
        avg_Q /= self.num_runs
        state_visits /= self.num_runs

        # Return the results
        return avg_Q, rewards, steps, state_visits
```

## 3.2 $Q-$Learning

### 3.2.1 What is $Q-$Learning?

$Q-$Learning is an off-policy[5] algorithm designed for solving MDPs[6]. It learns a policy by interacting with an environment, updating $Q-$values based on observed rewards and actions taken, and is not restricted to following the policy used for exploration.

### 3.2.2 Why is $Q-$Learning Used?

$Q-$Learning is used for:

- Off-policy learning, allowing the agent to explore independently of the policy being learned.
- Simplicity and ease of implementation.
- Learning optimal policies in Markov Decision Processes.

### 3.2.3 The Algorithm

**Algorithm 2** : $Q-$Learning Algorithm

Initialize $Q(s, a)$ arbitrarily
**for** each episode **do**
    Initialize $S$
    **while** $S$ is not the goal state and max steps not reached **do**
        Choose action $A$ from state $S$ using policy ($\epsilon-$greedy/ softmax) derived from $Q$
        Take action $A$, observe reward $R$ and next state $S'$
        $$\boxed{Q(S, A) \leftarrow Q(S, A) + \alpha \left[ R + \gamma \max_{a'} Q(S', a') - Q(S, A) \right]}$$
        $S \leftarrow S'$
    **end while**
**end for**

### 3.2.4 An Overview

**Initialization Phase**:

- $Q-$**Value Initialization**: Initialize $Q-$values for all state-action pairs arbitrarily.

**Exploratory Phase**:

- **Action Selection**: In each state, the agent selects an action using an epsilon-greedy policy for exploration and exploitation.
- **Action Execution and Environment Interaction**: The agent executes the chosen action, and the environment responds with a reward and transitions to a new state.

**Learning Phase**:

[5] exploration is almost independent of the policy being learned [6] Markov Decision Processes

- **Next Action Selection**: Based on the new state, the agent selects the next action using the epsilon-greedy policy.
- $Q-$**Value Update (**$Q-$**Learning)**: Update the $Q-$value of the current state-action pair using the $Q-$Learning update rule:

$$Q(S,A) \leftarrow Q(S,A) + \alpha \left[ R + \gamma \max_{a'} Q(S',a') - Q(S,A) \right]$$

**Repeat and Termination**:

- **Repeat Steps 2 and 3**: Continue exploration, action selection, and learning until reaching a terminal state or predefined iterations.
- **Termination**: The algorithm terminates when reaching a terminal state or after a predefined number of iterations.

### 3.2.5 The idea Behind $Q-$Learning

The fundamental idea is to estimate the expected cumulative reward of following an optimal policy in a given environment. $Q-$Learning achieves this by iteratively updating $Q-$values based on observed rewards and the maximum $Q-$value of subsequent state-action pairs.

### 3.2.6 Pros of $Q-$Learning

- **Off-Policy Flexibility**: $Q-$Learning is off-policy, providing flexibility in exploration independent of the policy being learned.
- **Simplicity**: $Q-$Learning is relatively simple and easy to implement.
- **Optimal Policy Learning**: With sufficient exploration, $Q-$Learning converges to the optimal policy.

### 3.2.7 Cons of $Q-$Learning

- **Exploration Challenges**: While off-policy, $Q-$Learning may still face challenges in environments requiring extensive exploration.
- **Sample Inefficiency**: It might require a large number of samples to converge.
- **Limited to Finite MDPs**: $Q-$Learning is designed for finite MDPs and may face challenges in continuous or large state and action spaces.

### 3.2.8 Implementation of $Q-$Learning

Under the assumption of the standard representation of Python libraries[7], analogous to the class `SARSA`, the following piece of Python code implements $Q-$Learning as an extension of the class `TDQLearner` as explained in Algorithm 2.

[7] Again, recall that usually the library `numpy` is imported as `np`

```
class QLearning(TDQLearner):
    def train(self):
        # Initialize arrays to store results
        rewards = np.zeros((self.num_runs, self.num_episodes))
        steps = np.zeros((self.num_runs, self.num_episodes))
        avg_Q = np.zeros((self.env.num_states, self.env.num_actions))
        state_visits = np.zeros((self.env.num_states, 1))

        # Loop over multiple runs
        for run in range(self.num_runs):
            # Initialize Q-values for each state-action pair
            Q = np.zeros((self.env.num_states, self.env.num_actions))

            # Loop over episodes within a run, using tqdm for progress visualization
            for episode in tqdm(range(self.num_episodes), desc="Run: "+str(run+1), position=1):
                # Reset the environment for a new episode
                state = self.env.reset()
                done = False

                # Continue until the episode is done
                while not done:
                    try:
                        # Choose an action using the Q-values and the exploration strategy
                        action = self._choose_action(Q[state, :])

                        # Take the chosen action and observe the next state and reward
                        next_state, reward = self.env.step(state, action)

                        # Update Q-value for the current state-action pair using the Q-learning
                        ↪  update rule
                        Q[state, action] += self.alpha * (reward + self.gamma *
                        ↪  np.max(Q[next_state, :]) - Q[state, action])

                        state = next_state # Move to the next state

                        # Record rewards, steps, and state visits
                        rewards[run, episode] += reward
                        steps[run, episode] += 1
                        state_visits[state] += 1

                        # Check if the goal state is reached
                        if state in self.env.goal_states_seq or steps[run, episode] >
                        ↪  max_steps:
                            done = True
                    except IndexError:
                        continue  # Ignore IndexErrors (in case of unexpected issues)

            # Accumulate Q-values for averaging
            avg_Q += Q

        # Average Q-values and state visits over all runs
        avg_Q /= self.num_runs
        state_visits /= self.num_runs

        # Return the results
        return avg_Q, rewards, steps, state_visits
```

## 3.3 A discussion on Policies

### 3.3.1 What is meant by a policy (here)?

In SARSA and $Q-$learning, a policy is a mapping from states to actions that guides the agent's decision-making process. The policy specifies what action the agent should take in a given state to maximize its expected cumulative reward. Both SARSA and $Q-$learning aim to learn an optimal policy by iteratively updating $Q-$values based on observed rewards and actions taken in the environment.

**SARSA**: SARSA is an *on-policy* algorithm, meaning that the policy used for exploration (choosing actions) is the same as the policy used for learning (updating $Q-$values). The agent's policy directly influences its actions during training.

**$Q-$learning**: $Q-$learning, on the other hand, is an *off-policy* algorithm. It explores independently of the policy being learned, allowing the agent to learn from a mix of exploration and exploitation experiences. The policy implicit in $Q-$values guides action selection during exploration.

### 3.3.2 Exploration Strategies

#### 3.3.2.1 The Softmax

Given $Q$ and $\tau$, the softmax policy converts the $Q-$values into a probability distribution over actions, allowing the agent to probabilistically choose actions based on their associated Q-values; in other words, it selects an action $A$ with a probability:

$$\boxed{P(A=a)=\frac{\exp\left(\frac{Q(a)}{\tau}\right)}{\sum_i \exp\left(\frac{Q(i)}{\tau}\right)}}$$

where $\tau$ is the temperature parameter controlling the level of exploration. To prevent potential errors caused by exploding values, $Q$ is scaled before applying the softmax.

#### 3.3.2.2 The $\epsilon-$greedy

Given $Q$ and $\epsilon$, the $\epsilon$-greedy policy balances exploration and exploitation by choosing the action with the highest $Q-$value with probability $1-\epsilon$ (aka exploitation) and selecting a random action with probability $\epsilon$ (aka exploration); in other words, it selects:

$$\boxed{A=\begin{cases}\text{An arbitrary action,} & \text{with probability } \epsilon \\ \text{The action with maximum } Q, & \text{with probability } 1-\epsilon\end{cases}}$$

where $\epsilon$ is the exploration rate.

# 4 Problem 2

For each algorithm, run experiments for the start states $(0, 4)$ and $(3, 6)$. Consider the following three variants of stochasticity for each start state.

1. `wind=False` (clear), $p = 1.0$ (deterministic step).
2. `wind=False` (clear), $p = 0.7$ (stochastic step).
3. `wind=True` (windy), $p = 1.0$ (deterministic step).

In total, you have to conduct 12 different experiments – (3 grid world variants $\times$ 2 start states $\times$ 2 algorithms).

---

## 4.1 A discussion on the Start states

We have two start states, namely $(0, 4)$ and $(3, 6)$, considered separately for an experiment.

For the start state $(0, 4)$, there exists at least one optimal path to each of the goal states with the best possible reward being $-6$ listed below. Note that, for a cell $(i, j)$, we represent it by $10i + j$. So the start state is 4 and the goal states are – 22, 9 and 87.

1. To reach goal 22 with the best reward, the set of optimal paths is precisely the one with the minimum number of steps, there is one such unique path.
2. To reach goal 9, with the best reward, take any shortest path from 4 to 9 that avoids the restart state at 37 and the bad state at 19.
3. To reach goal 87, with the best reward, take any shortest path from 4 to 87 that avoids the bad state at 44 and 75.

For the start state $(3, 6)$, there exists at least one optimal path to each of the goal states $(0, 9)$ and $(8, 7)$ with the best possible reward being $-1$, but for $(2, 2)$ the best possible reward is $-12$. Note that, for a cell $(i, j)$, we represent it by $10i + j$. So the start state is 4 and the goal states are – 22, 9 and 87.

1. To reach goal 22 with the best reward, the set of optimal paths is precisely the one with the minimum number of steps.
2. To reach goal 9, with the best reward, take any shortest path from 4 to 9 that avoids the restart state at 37 and the bad state at 19.
3. To reach goal 87, with the best reward, take any shortest path from 4 to 87 that avoids the bad state at 75.

## 4.2 A discussion on the stochasticity variants

We have three possibilities for the stochasticity.

1. `wind=False` (clear), `p=1.0` (deterministic step): This represents a deterministic grid with an agent whose behaviour is deterministic to take the direction it is pointing at (the relative north) and 0 to the relative east and relative west cell.

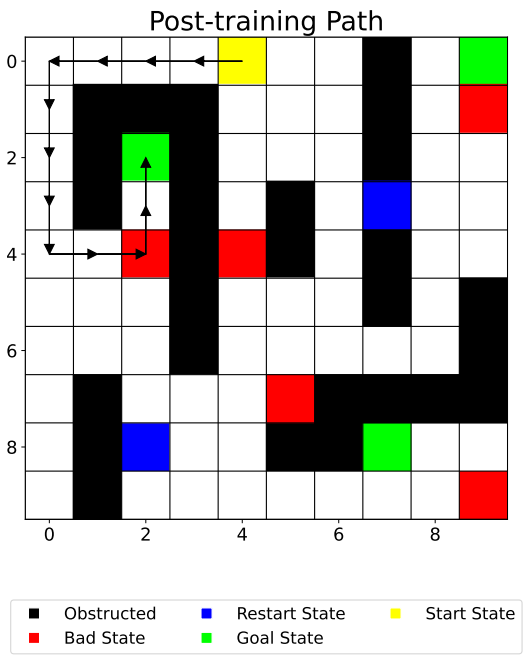
2. `wind=False` (clear), `p=0.7` (stochastic step): This represents a deterministic grid with an agent whose behaviour is probabilistic. It takes the direction it is pointing at (the relative north) with a probability of 0.7 and with a probability of 0.15 it takes either of the relative east and relative west direction.[8]

3. `wind=True` (clear), `p=1.0` (deterministic step): This represents a stochastic grid where at each time step with a probability of 0.4 the wind is blowing from the absolute west to the absolute east. The behaviour of the agent in this case is deterministic, which is the same as the first variant. It takes the direction it is pointing at (the relative north) with a cent percent certainty and with a probability 0 it takes either of the relative east and relative west direction. Note that 0.4 is quite significant and in some pairs of (Policy, Algorithm), the agent effectively learns to use the wind to its advantage.

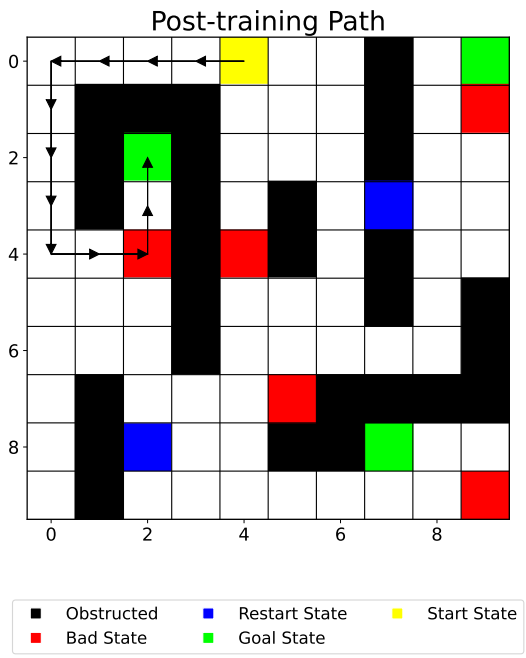
## 4.3 Paths taken after training

For each algorithm, we ran the experience for each of the variants of the stochasticity for each of the start states – $(0, 4)$ and $(3, 6)$ and for each of the policies – softmax and $\epsilon-$greedy with the best set of hyperparameters (after tuning)[9]. The following plots display the path taken after the completion of the training over 5K episodes. For each case, we have plotted the optimal policy path for each of the 12 experiments (that is – for each experiment, we have plotted the one which gave a better reward between softmax (default) and $\epsilon-$greedy).

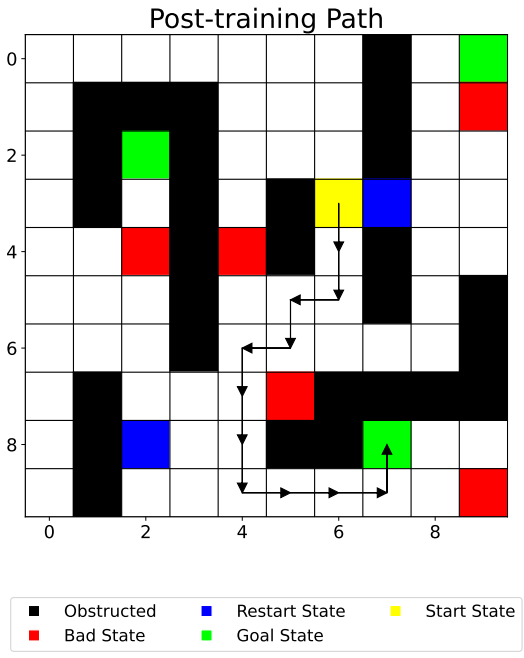
[8] For each of the experiments we consider the bias ($b$) as 0.5. [9] Check out Problem 3.



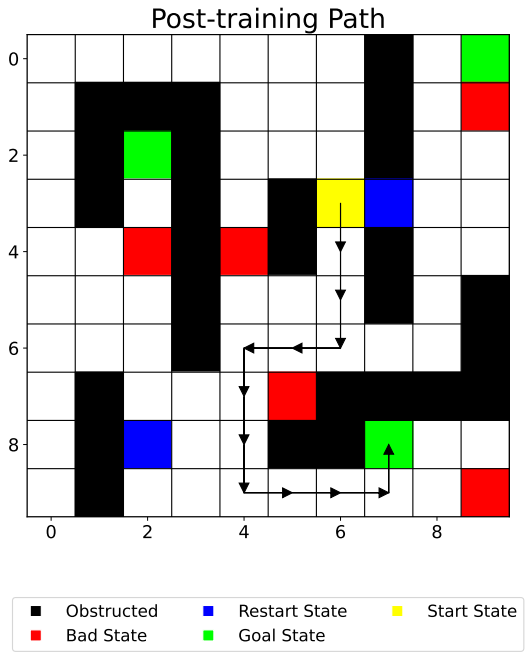

(a) Start state: $[0, 4]$; Algorithm: SARSA




(b) Start state: $[0, 4]$; Algorithm: Q-Learning

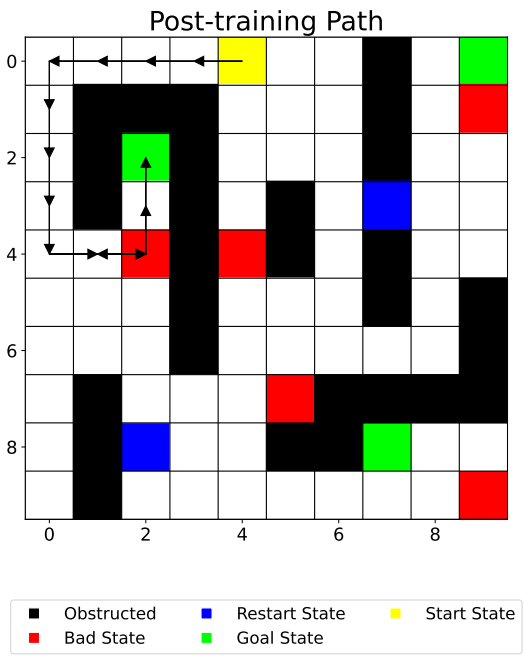



(c) Start state: $[3, 6]$; Algorithm: SARSA

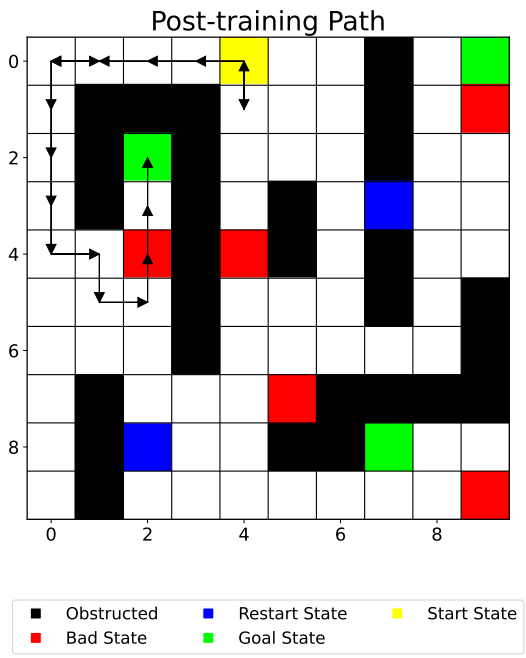



(d) Start state: $[3, 6]$; Algorithm: Q-Learning

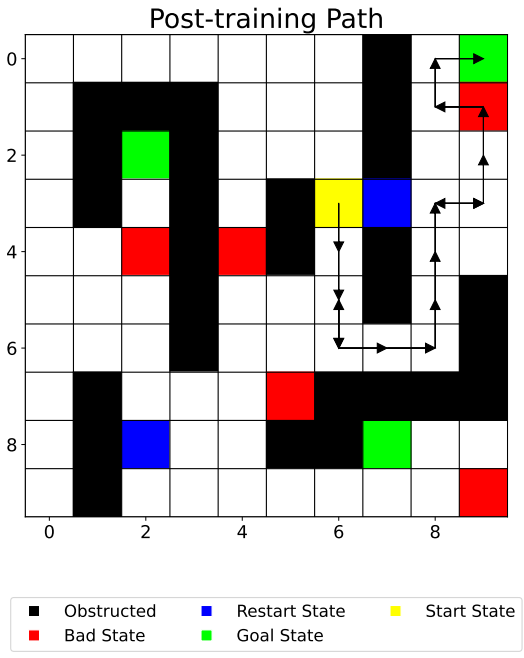
Figure 2: Stochasticity Variant 1 : wind=False(clear), $p = 1.0$



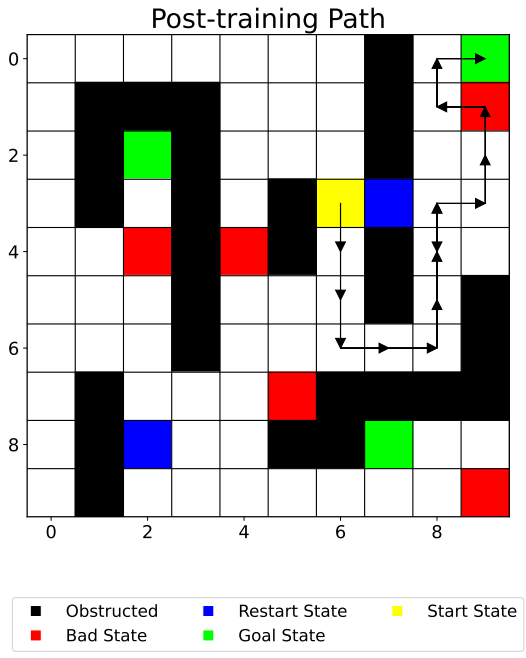

(a) Start state: $[0, 4]$; Algorithm: SARSA




(b) Start state: $[0, 4]$; Algorithm: Q-Learning

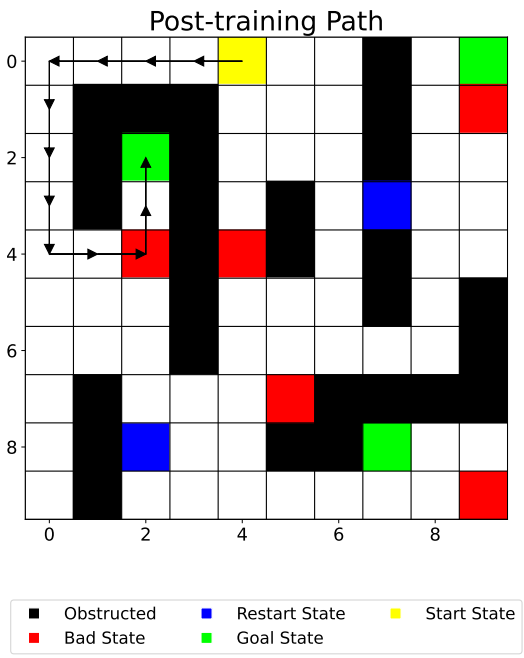



(c) Start state: $[3, 6]$; Algorithm: SARSA

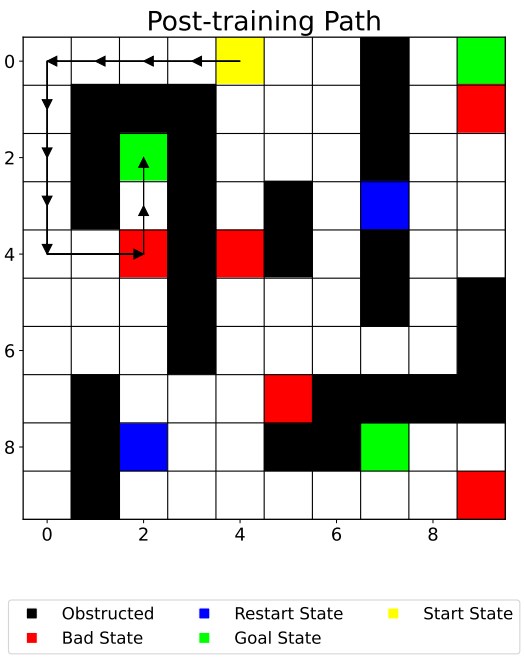



(d) Start state: $[3, 6]$; Algorithm: Q-Learning

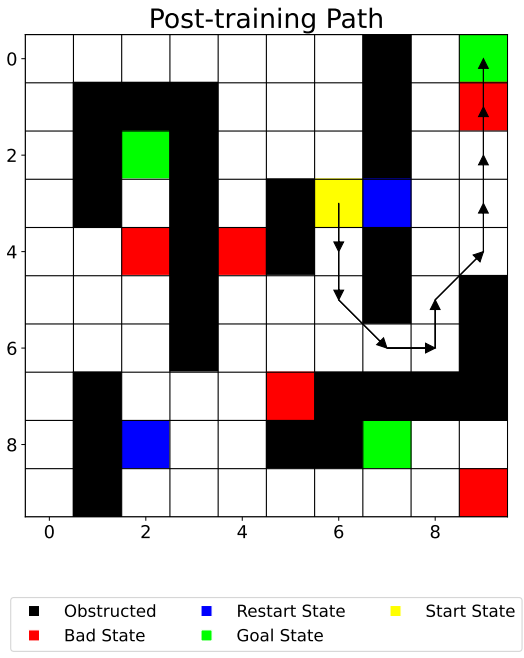
Figure 3: Stochasticity Variant 2 : wind=False(clear), $p = 0.7$, $b = 0.5$



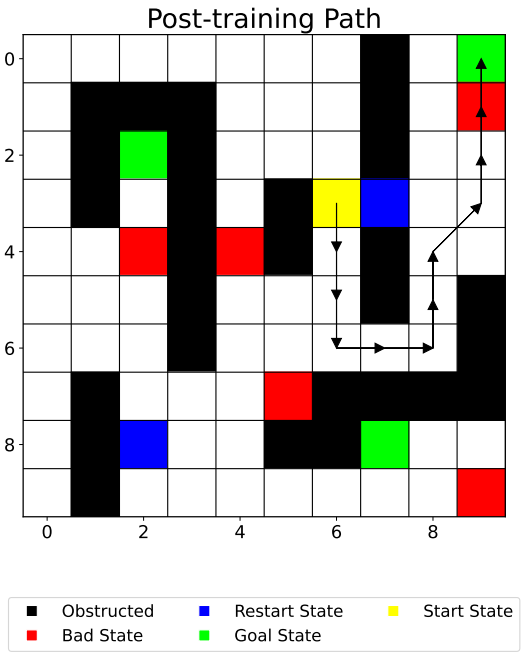

(a) Start state: $[0, 4]$; Algorithm: SARSA




(b) Start state: $[0, 4]$; Algorithm: Q-Learning




(c) Start state: $[3, 6]$; Algorithm: SARSA




(d) Start state: $[3, 6]$; Algorithm: Q-Learning

Figure 4: Stochasticity Variant 3 : wind=True(clear), $p = 1.0$

## 4.4 Inference on the post-training path

We note down some inferences corresponding to the path taken by the agent post-training best out over $\epsilon-$greedy and softmax. Each of them actually represents the softmax variants and it turns out that Softmax is quite better as compared to $\epsilon-$greedy in each of the experiments. The reader may find out the reason in Problem 4.

1. **Stochasticity variant 1:** For the non-windy deterministic step, the agent always learns the optimal path and traces it post-training in each of SARSA and $Q-$Learning, which can be seen in Figure 2.

2. **Stochasticity variant 2:** For the nonwindy stochastic step in Figure 3, the behaviour of the agent is quite interesting. It might be noticed in each of the four variants (Start state, Algorithm), that sometimes the agent ends up retracing some earlier visited portions of the path.

   Also, we observe a peculiar behaviour by the agent for the start state (3, 6), that is the start state indexed at 36. Note that after reaching at cell (3, 8), aka cell 38, it prefers immediately going to (3. 9) as compared to (2, 8) even though the latter has a straightforward path till the goal state (0, 9) with the optimal reward. It is because there is a restart state at (3, 7). Standing at (3, 8), if the agent moves to (2, 8), that is the relative north, there is 0.15 probability that it will end up in the restart state with a tremendous penalty of $-100$ and ending up in the start state again, whereas if we head to (3, 9) there is 0 probability to end up in a restart state and each of the states we immediately end up at, is a normal state with a $-1$ penalty. The credit of visiting the bad state at (1, 9) from (2, 9) can be given to the stochasticity nature of the agent.

3. **Stochasticity variant 3:** Now comes the windy determinstic path in Figure 4. When the start state is (0, 4), the agent ends up going to (2, 2), but the path taken is longer[10], for eg – for the SARSA one the path is as follows – `[4, 3, 2, 1, 1, 1, 0, 10, 20, 30, 40, 41, 42, 32, 22]`, which is a suboptimal path with a reward of -8, whereas in $Q-$Learning it takes the path `[4, 3, 2, 2, 2, 1, 0, 10, 20, 30, 40, 42, 32, 22]` with a reward of -7 instead of -8. Note that it uses the wind to directly form 40 to 42.

   For the start state of (3, 6), the agent takes the path to (0, 9) as it is towards the absolute west of the starting state as compared to the other two goal states, one of which is in the south and the other in the west. Note that it uses the wind to go from (5, 6) to (6, 7) avoiding (5, 6) in SARSA. Also, it does not take the optimal path from (5, 8) in SARSA and (4, 8) in $Q-$Learning, as the optimal path is to maintain the column till (0, 8) and then take right to (0,9). The probability that wind does not blow while going from (5, 8) to (0, 8) is $(0.4)^5 \approx 0.01$, which is negligible. Hence it ends up getting drifted to the last column in between its journey.

[10] Please visit the same plot in Problem 3

# 5 Problem 3

For each of the 12 experiments, determine the best set of hyperparameters($\tau$ in softmax or $\varepsilon$ in $\varepsilon$-greedy, learning rate $\alpha$, and discount factor $\gamma$), along with the best action selection policy($\varepsilon$-greedy or softmax) with justification. You are also required to plot the following (4 plots per exp) for each experiment(with the best hyperparameters and action selection policy):

1. Reward curves and the number of steps to reach the goal in each episode.
2. Heatmap of the grid with state visit counts, i.e., the number of times each state was visited throughout the training phase.
3. Heatmap of the grid with $Q$ values after training is complete, and optimal actions for the best policy.

---

## 5.1 A discussion on hyperparameters

### 5.1.1 Learning Rate ($\alpha$)

1. **Definition:** The learning rate determines the step size at which the agent updates its Q-values or policy based on new information. It controls how much the agent adjusts its current knowledge in response to new experiences.
2. **Significance:** A higher learning rate allows the agent to adapt quickly to recent information, but it may lead to instability. A lower learning rate results in slower but potentially more stable learning.
3. **Range:** Common values are between 0 and 1. Smaller values (e.g., 0.1) often lead to stable learning.

### 5.1.2 Discount Factor ($\gamma$)

1. **Definition:** The discount factor influences how much the agent values future rewards. It determines the extent to which the agent considers long-term consequences in decision-making.
2. **Significance:** A higher discount factor (closer to 1) encourages the agent to prioritize long-term rewards, fostering farsighted decision-making. A lower discount factor (closer to 0) makes the agent focus more on immediate rewards.
3. **Range:** Typically between 0 and 1.

### 5.1.3 Exploration-Exploitation Tradeoff ($\epsilon$)

1. **Definition:** $\epsilon$ is used in exploration strategies, like epsilon-greedy. It controls the probability that the agent selects a random action instead of the one with the highest estimated value.
2. **Significance:** A higher $\epsilon$ promotes exploration, allowing the agent to discover new strategies. A lower $\epsilon$ emphasizes exploitation, where the agent relies more on its current knowledge.
3. **Decay:** $\epsilon$ values often start high and decay over time to shift from exploration to exploitation with providing more weightage to the latter. But in this project, it has been taken constant over time, which facilitates the equivalent exploration even after a large number of episodes.

### 5.1.4 Softmax Temperature ($\tau$)

1. **Definition:** $\tau$ is used in softmax action selection. It controls the level of randomness in the agent's choice of actions. A higher $\tau$ leads to more randomness, while a lower $\tau$ results in more deterministic actions.

2. **Significance:** Higher $\tau$ values introduce more randomness, encouraging exploration. Lower $\tau$ values make the agent's actions more deterministic, favouring the exploitation of known strategies.

3. **Decay:** Similar to $\epsilon$, $\tau$ is often decayed over time to balance exploration and exploitation with providing more weightage to the latter. But in this project, it has been taken constant over time, which facilitates the equivalent exploration even after a large number of episodes.

These hyperparameters play crucial roles in shaping the behaviour of the agent and can significantly impact the convergence and performance of the learning process. Fine-tuning these hyperparameters is often necessary to achieve optimal results in different environments and tasks. The details regarding these fine-tunings have been discussed in Problem 4.

Too much generalized writing! Now, let's focus on some problem-specific questions.

## 5.2 Values considered for the hyperparameter tuning

We trained each of the 24 experiments (3 stochasticity variants $\times$ 2 start states $\times$ 2 algorithms $\times$ 2 policies), from the set of following hyperparameters:

```
alphas = [0.10, 0.20, 0.30]
gammas = [0.90, 0.95, 0.99]
epsilons = [0.01, 0.05, 0.10]
taus = [0.01, 0.05, 0.10]
```

Followingly, we discuss the reason behind considering such a small set of hyperparameters for the final tuning purpose.

1. As it comes to the learning rate $\alpha$, it goes to show that higher values of $\alpha$ (typically $\geqslant 0.6$, sometimes $\sim 0.4$) lead to oscillations or overshooting, where the learning process becomes unstable and 5K episodes are sometimes not enough to train the model; in other words, convergence to an optimal policy becomes difficult due to the constant and large updates to the agent's knowledge. Also, high learning rates make the algorithm more sensitive to noisy and outlier experiences, such as – the windy environment or the agent behaving in a stochastic manner (a non-unity $p$), potentially causing the agent to overreact to outliers.

2. Concerning the discount factor $\gamma$, observe that for the final hyperparameter-tuning, we consider only the values $\geqslant 0.9$. Note that, while training with $\gamma$ being 0.85, or even 0.8, we observe some unexpected behaviours. A low $\gamma$ places less importance on future rewards, making the agent focus more on immediate rewards, which is significant as the environment involves delayed rewards or rewards that occur in the distant future (for e.g reaching the goal state provides a reward of $+10$), a low $\gamma$ leads the agent to undervalue such rewards, thus developing a short-sided learning attitude.

3. Let's talk about the parameter $\epsilon$. A larger value of $\epsilon$, such as $\geqslant 0.2$ (that is – there is a 20% chance that the agent will explore even after 5K episodes), means the agent frequently selects random actions. While this encourages exploration, it can lead to suboptimal performance, as the agent may not exploit its current knowledge effectively. A low value of $\epsilon$ represents the fact that the

agent will almost always choose the action with the highest estimated value. While this may exploit the current knowledge effectively, it limits the agent's ability to explore new actions and discover potentially better strategies. Thus an intermediate value range for $\epsilon$ is chosen. Also, since we are focusing on a time-invariant $\epsilon$[11], it restricts us to use values only at a very narrow range of 0.01 to 0.1.

4. Analogously, as far as the value of the temperature parameter $\tau$ is concerned, note that for a larger value of $\tau$, the exponentiation term $\left(\frac{Q(a)}{\tau}\right)$ becomes smaller, and the softmax probabilities become more uniform, that encourages more exploration; whereas $\tau$ decreases, the softmax probabilities become more focused on the action with the highest value. Also, we restrict our attention to time-invariant $\tau$, which consequently guides us to the narrowed-down range of $0.01 - 0.1$ as the optimal consideration for the temperature parameter.

5. The reader may also verify that a small perturbation of the hyperparameter value within the reasonably allowed range[12] does not impact the final big picture to a noticeable extend. Thus, the least count we consider is 0.1 for $\alpha$, 0.05 for $\gamma$ and 0.05 for each of $\epsilon$ and $\tau$.

6. Note that, we have trained each of the experiments ONLY till $5K$ episodes. It is always possible that a higher value of the number of episodes may favour a different set of hyperparameters for tuning.

7. Also, observe that we have NOT assumed the hyperparameters to be independent as we have cross-trained with a 9 set of possibilities of $(\alpha, \gamma, \tau)$ for the softmax and $(\alpha, \gamma, \epsilon)$ for the $\epsilon-$greedy for each of the 24 individual experiments.

## 5.3 What is meant to be the best set of hyperparameters?

We must define what it means that a set of hyperparameters is the best, given this small set of possibilities as it determines the way we tune them. But before that, we must ask what is our objective. In other words, how we decide a set of hyperparameters outperforms another one, for the provided environment and the (Policy, Algorithm) pair. Let's talk about that.

With each of SARSA and $Q-$Learning, in our discussion, we aim to maximize the final reward as the agent's experience. Hence, the length of the final path taken by the agent is out of the discussion when it comes to capturing the best hyperparameter for a given (Policy, Algorithm). A path might be shorter, but if it leads to less reward, it is not desirable. So, the best set of hyperparameters provided must maximize the total reward. Okay, but is it, to maximize the maximum reward over all the $5K$ episodes averaged over the 5 runs, or is it to maximize the average reward over all the $5K$ episodes averaged over the 5 runs?

Note that if the situation is non-windy deterministic, such as the stochastic variant 1 and we have a pair of set of hyperparameters, we must ask whether each one of them reaches the best possible reward that can be achieved, which is the answer to the former question above. And if both of them have the same maximum possible reward, we should look at which one converged faster; also this addresses the question of which one has generated the better reward on average and this is the answer to the latter question asked in the last paragraph. But note that for a purely non-windy deterministic environment, once an agent learns the optimal path, it does not tend to go through other paths because of its/ environment's stochasticity (but it is possible if the set of hyperparameters is not good enough). Also, since 5k is a reasonably large enough episode (that usually experiences suboptimal reward for non-windy deterministic cases for the initial 200 episodes), the maximum reward gets subsumed within the average reward gained.

[11] that is – $\epsilon$ at episode #1 is the same as that at episode #5K. [12] For e.g. – We should not consider $\gamma \leqslant 0$ at least for this experiment

Now, let's talk about when the environment/ agent's behaviour itself is stochastic. For a set of hyperparameters if a (Policy, Algorithm) generates the highest possible maximum reward over 5K episodes (averaged over 5 runs), due to the stochasticity it may not generate the same reward post-training for the same set of hyperparameters. So, for each of the 24 cases we considered a set of hyperparameters optimal if it generates a maximum average reward over 5K episodes (which is averaged over 5 runs).

Enough of the theory! Now, let's focus our attention on the most interesting part, aka the plots. But before that, let's look at the plot overview.

## 5.4 The description of the plots

Each of the next 24 pages that the reader is going to look at, represents each of the 24 experiments run at the best possible hyperparameters (specific to each of them). Each of the pages consists of a table and 5 plots, which have been introduced below.

1. **The table:** It captures all the details of the parameters – wind parameter, the transition parameter ($p$), the bias parameter ($b$), starting position, algorithm and policy; the details of the hyperparameters – learning rate ($\alpha$), discount factor ($\gamma$), temperature factor ($\tau$) and exploration trade-off factor ($\epsilon$). It also shows the reward achieved through one of the paths taken post-training (shown in the figure next to the table) and the index sequence of the path. Note that, this index-sequencing represents each cell by a unique number between 0 and 99, that is for a cell $(i, j)$ we represent it by the number $10i + j$.

2. **The post-training path:** It is one of the paths taken post-training. Observe that if the reader runs the experiment to obtain a path post-training, it may generate the same precise path shown for the non-windy deterministic case (that is – the stochasticity variant 1), but may not generate the same path for the other two variants because of the stochasticity involved.

3. **The reward curve:** It is a plot of the reward (mean reward averaged over 5 runs in blue $\pm$ its standard deviation in light blue over 5 runs) against the episodes (0 - 5K).

4. **Steps to reach the goal:** Analogously, it is the plot of the number of steps taken from the initial start state to reach the goal state, aka the path length to the goal state (mean number of steps in red $\pm$ its standard deviation in light red over 5 runs) against the episodes (0 - 5K).

5. **Heat map of the state visit:** It represents the number of times a particular state cell visited over the 5K episodes averaged over 5 runs.

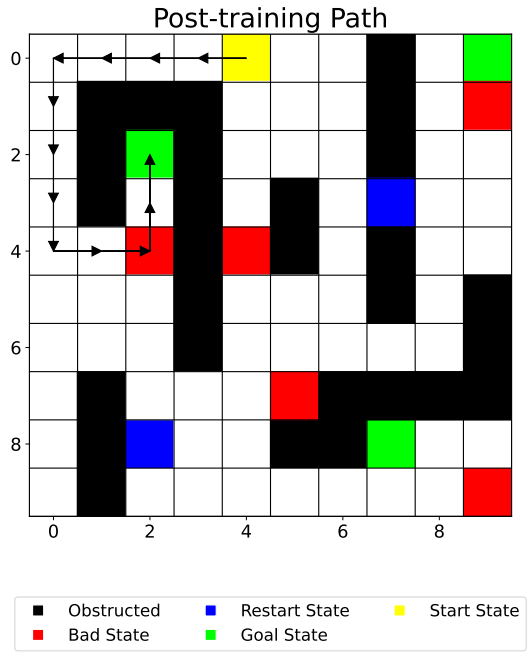
6. $Q-$ **value and optimal action plot:** It represents the heat map of $Q$ learnt through the given (Policy, Algorithm) for the provided environment and parameters averaged over 5 runs, the higher the value in the colour bar, so is the value of $Q$ (expected reward). A cross ($\times$) in this figure represents an obstructed state, and each of the arrows represents the optimal action at the particular state.

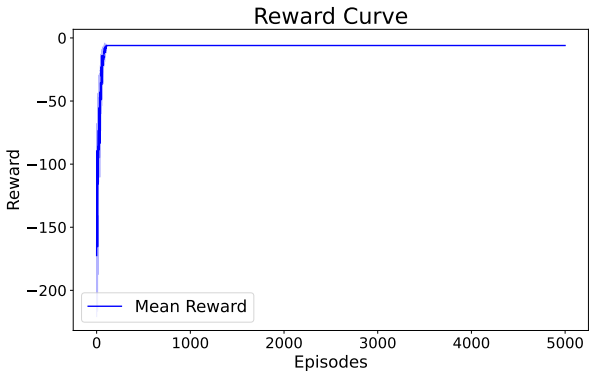
Here come the plots. Let's go!

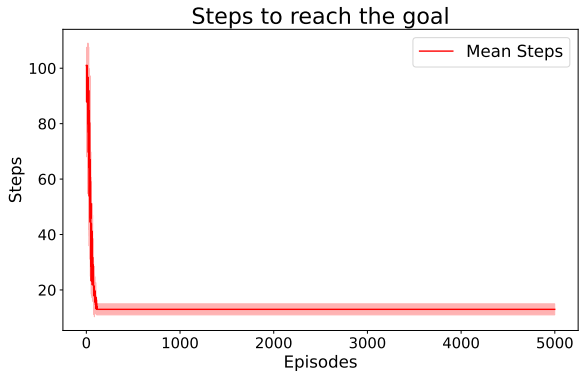
| Variables | Value |
|---|---|
| Wind parameter | `wind=False(clear)` |
| Transition Parameter ($p$) | 1.0 |
| Bias Parameter ($b$) | 0.5 |
| Starting Position | $(0, 4)$ |
| Algorithm | SARSA |
| Policy | Softmax |

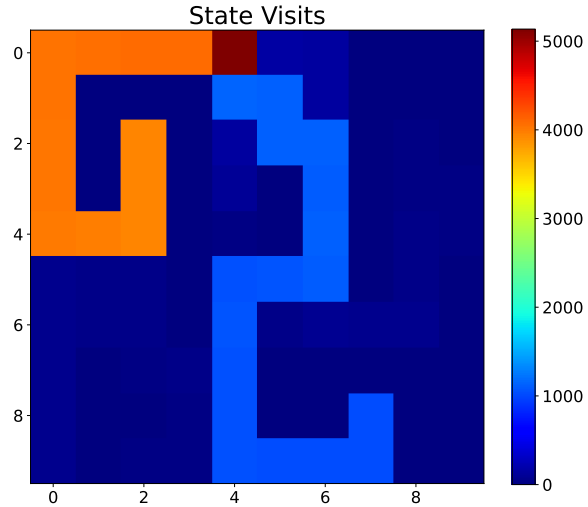
| Hyperparameter | Value |
|---|---|
| Learning rate ($\alpha$) | 0.30 |
| Discount factor ($\gamma$) | 0.99 |
| Temperature ($\tau$) | 0.05 |

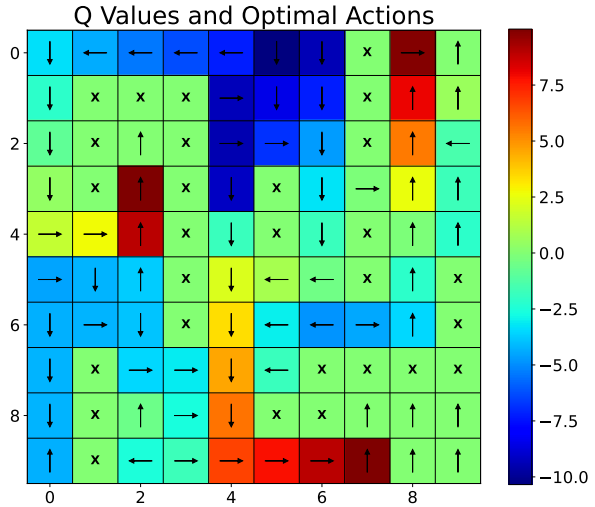
| | |
|---|---|
| Reward | -6 |
| Path taken | [4, 3, 2, 1, 0, 10, 20, 30, 40, 41, 42, 32, 22] |



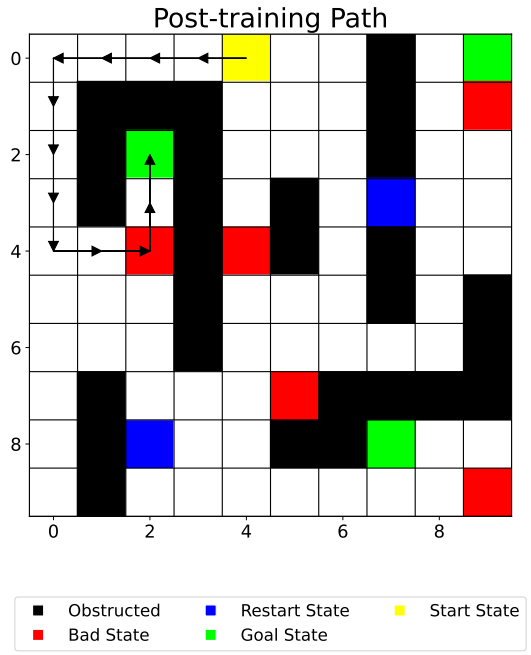



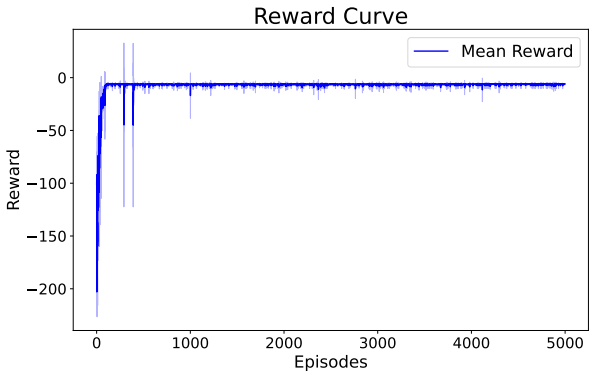



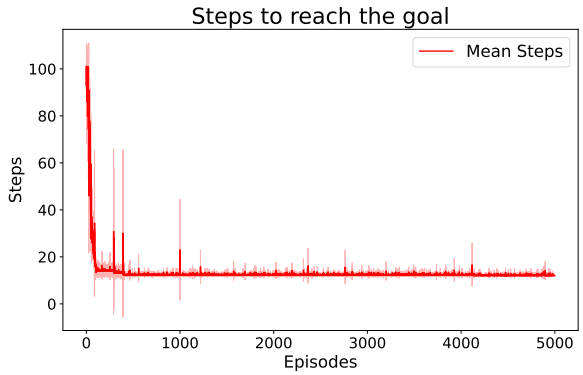
| Variables | Value |
|---|---|
| Wind parameter | `wind=False(clear)` |
| Transition Parameter ($p$) | 1.0 |
| Bias Parameter ($b$) | 0.5 |
| Starting Position | $(0, 4)$ |
| Algorithm | SARSA |
| Policy | $\epsilon-$greedy |

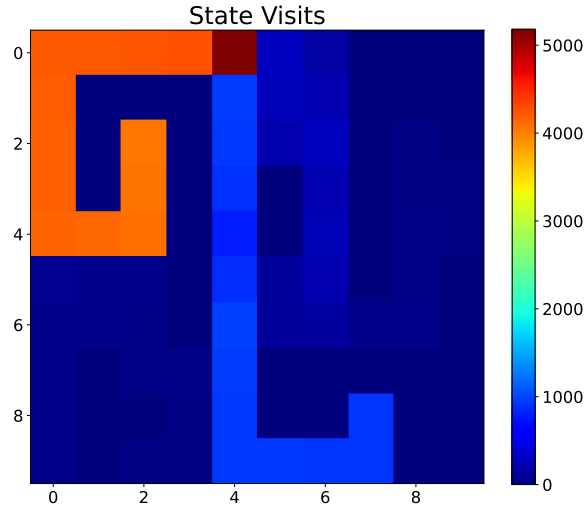
| Hyperparameter | Value |
|---|---|
| Learning rate ($\alpha$) | 0.30 |
| Discount factor ($\gamma$) | 0.95 |
| Epsilon ($\epsilon$) | 0.01 |

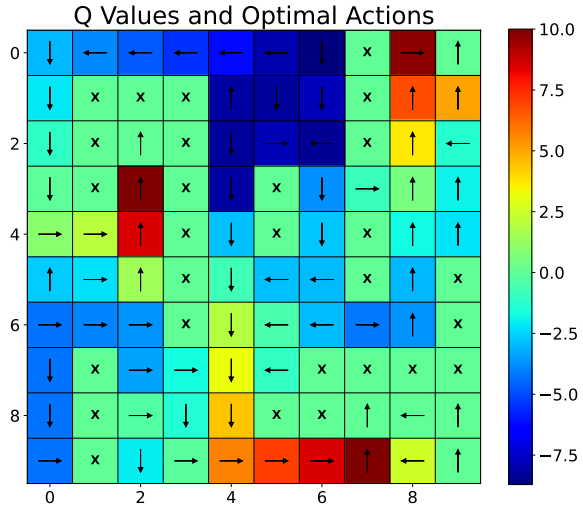
| | |
|---|---|
| Reward | -6 |
| Path taken | [4, 3, 2, 1, 0, 10, 20, 30, 40, 41, 42, 32, 22] |



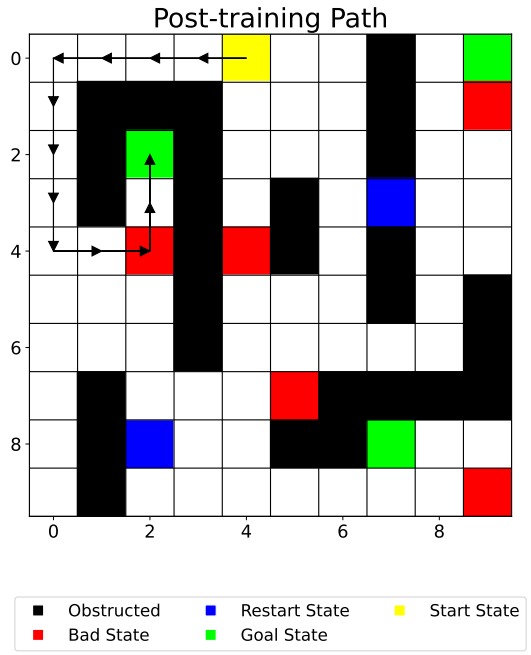



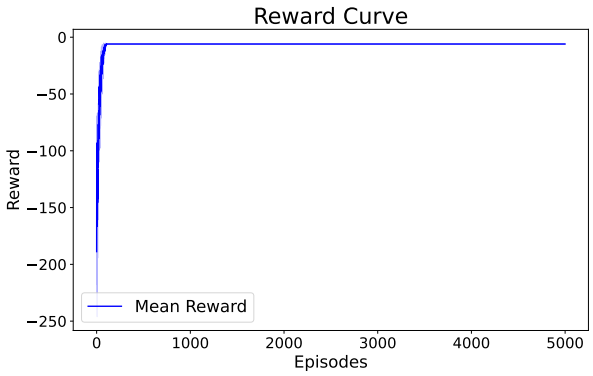



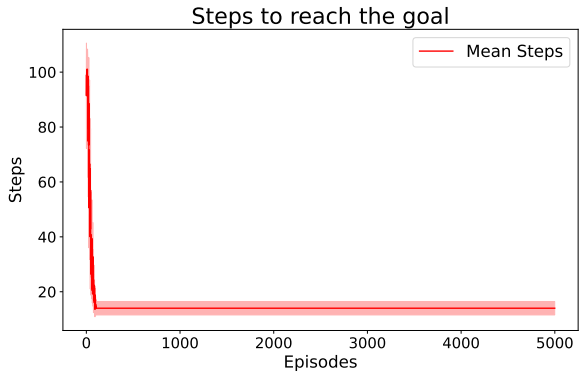
| Variables | Value |
|---|---|
| Wind parameter | `wind=False(clear)` |
| Transition Parameter ($p$) | 1.0 |
| Bias Parameter ($b$) | 0.5 |
| Starting Position | $(0, 4)$ |
| Algorithm | Q-Learning |
| Policy | Softmax |

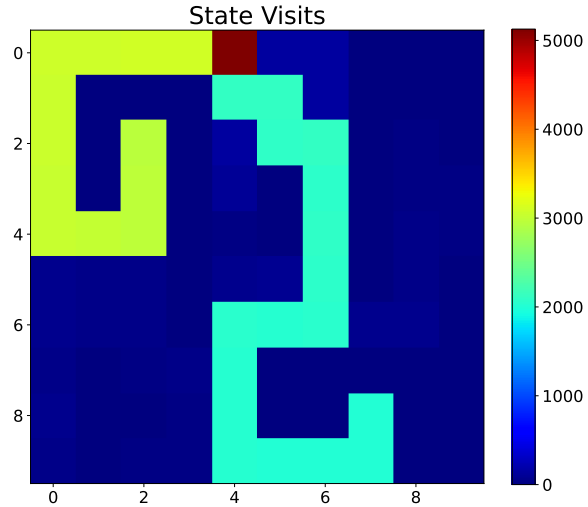
| Hyperparameter | Value |
|---|---|
| Learning rate ($\alpha$) | 0.30 |
| Discount factor ($\gamma$) | 0.99 |
| Temperature ($\tau$) | 0.01 |

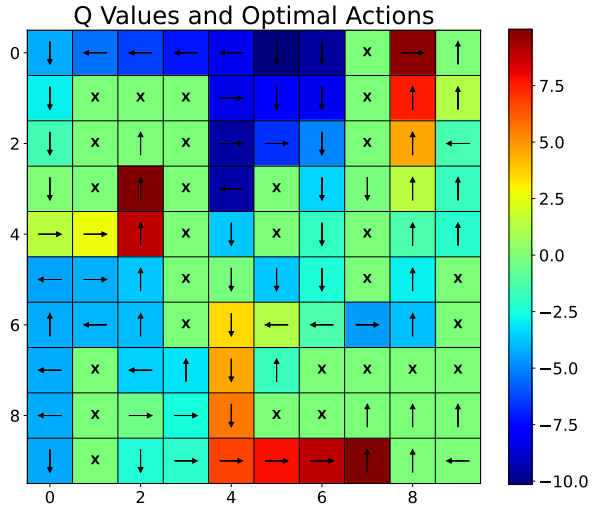
| | |
|---|---|
| Reward | -6 |
| Path taken | [4, 3, 2, 1, 0, 10, 20, 30, 40, 41, 42, 32, 22] |



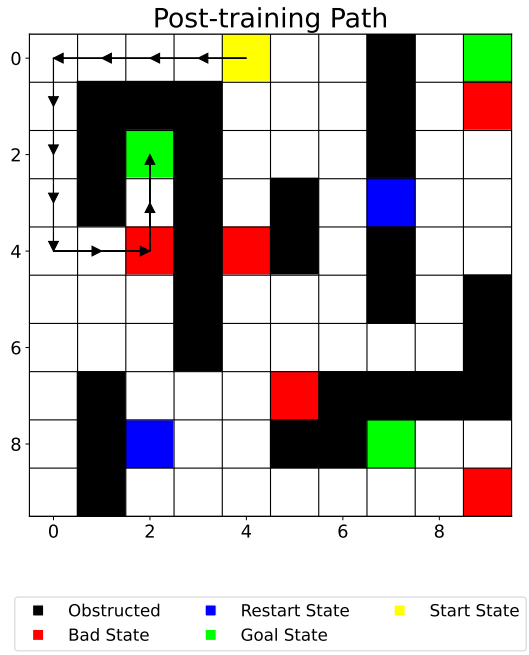



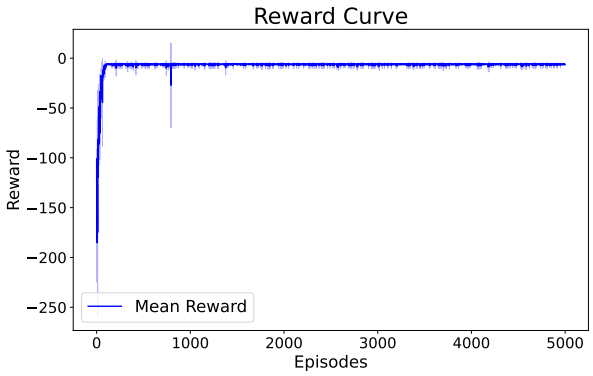



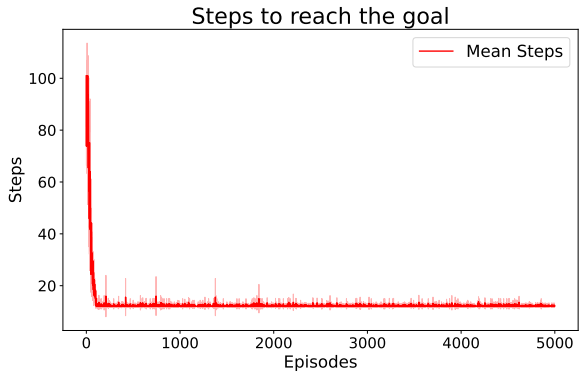
| Variables | Value |
|---|---|
| Wind parameter | `wind=False(clear)` |
| Transition Parameter ($p$) | 1.0 |
| Bias Parameter ($b$) | 0.5 |
| Starting Position | $(0, 4)$ |
| Algorithm | Q-Learning |
| Policy | $\epsilon-$greedy |

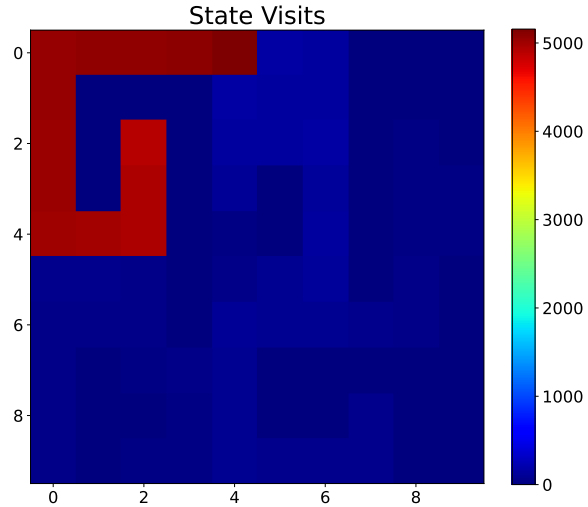
| Hyperparameter | Value |
|---|---|
| Learning rate ($\alpha$) | 0.30 |
| Discount factor ($\gamma$) | 0.99 |
| Epsilon ($\epsilon$) | 0.01 |

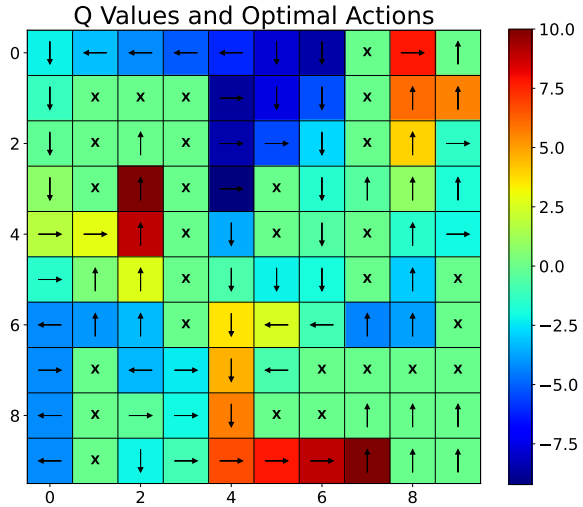
| | |
|---|---|
| Reward | -6 |
| Path taken | [4, 3, 2, 1, 0, 10, 20, 30, 40, 41, 42, 32, 22] |



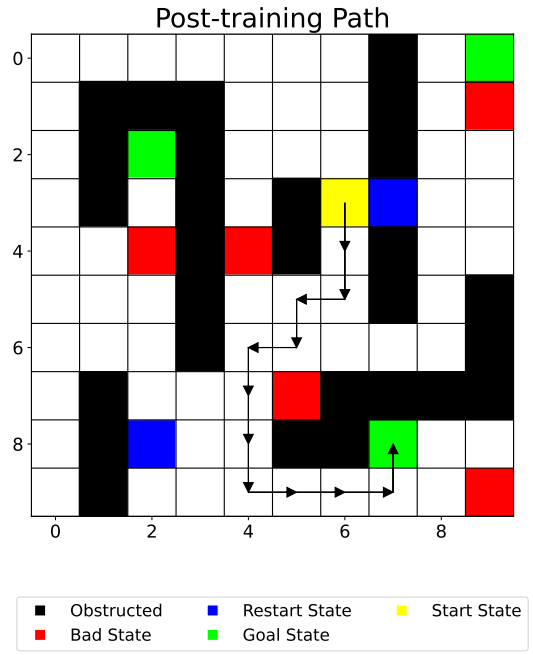



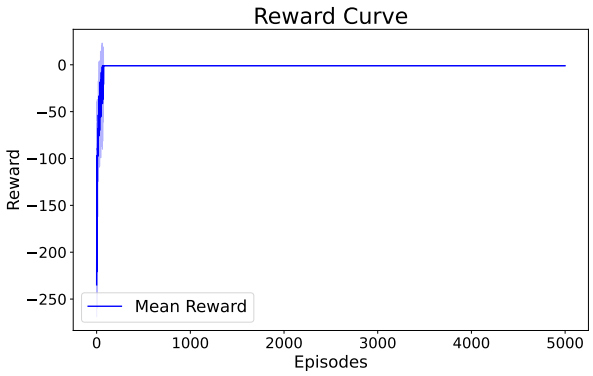



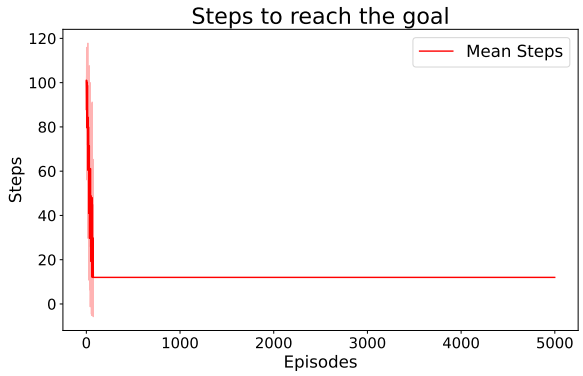
| Variables | Value |
|---|---|
| Wind parameter | `wind=False(clear)` |
| Transition Parameter ($p$) | 1.0 |
| Bias Parameter ($b$) | 0.5 |
| Starting Position | $(3, 6)$ |
| Algorithm | SARSA |
| Policy | Softmax |

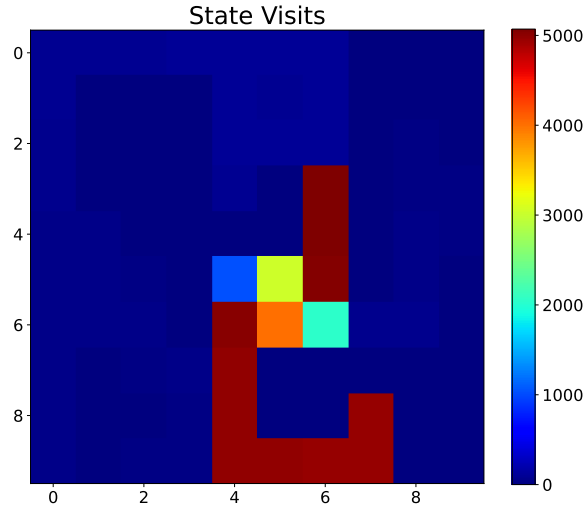
| Hyperparameter | Value |
|---|---|
| Learning rate ($\alpha$) | 0.30 |
| Discount factor ($\gamma$) | 0.99 |
| Temperature ($\tau$) | 0.01 |

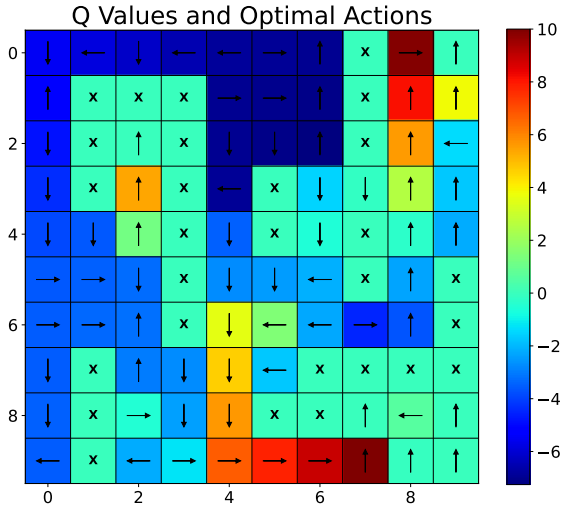
| | |
|---|---|
| Reward | -1 |
| Path taken | [36, 46, 56, 55, 65, 64, 74, 84, 94, 95, 96, 97, 87] |



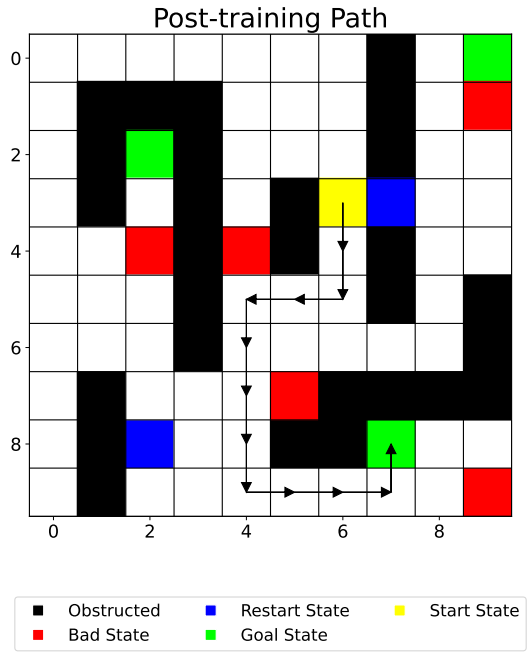



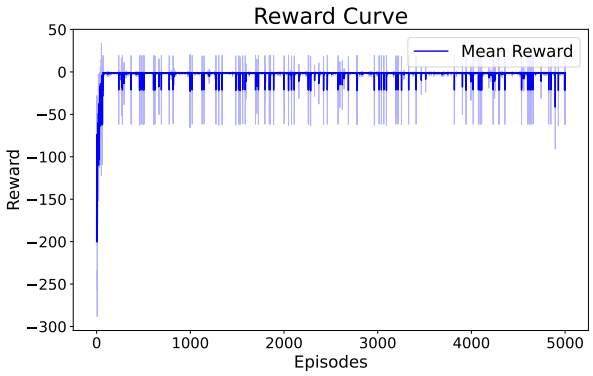



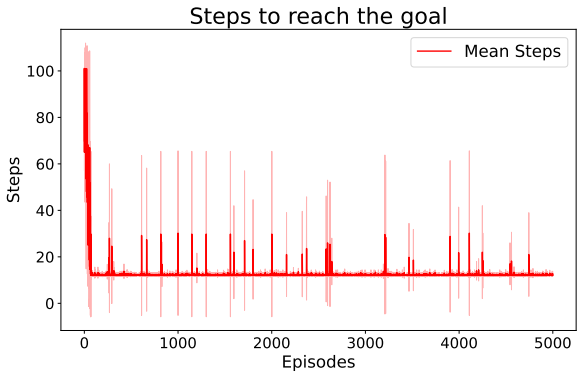
| Variables | Value |
|---|---|
| Wind parameter | `wind=False(clear)` |
| Transition Parameter ($p$) | 1.0 |
| Bias Parameter ($b$) | 0.5 |
| Starting Position | $(3, 6)$ |
| Algorithm | SARSA |
| Policy | $\epsilon-$greedy |

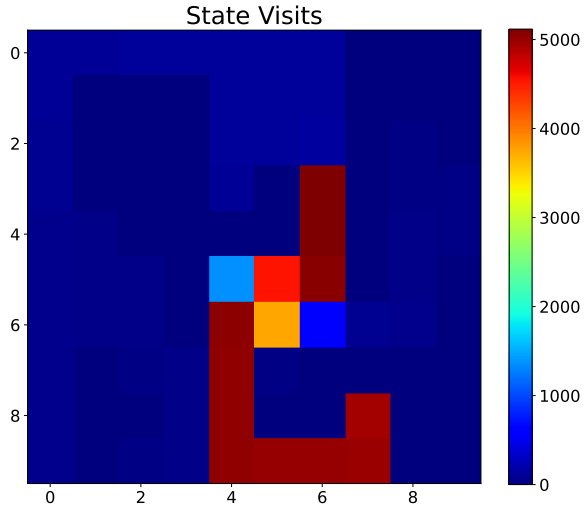
| Hyperparameter | Value |
|---|---|
| Learning rate ($\alpha$) | 0.30 |
| Discount factor ($\gamma$) | 0.90 |
| Temperature ($\tau$) | 0.01 |

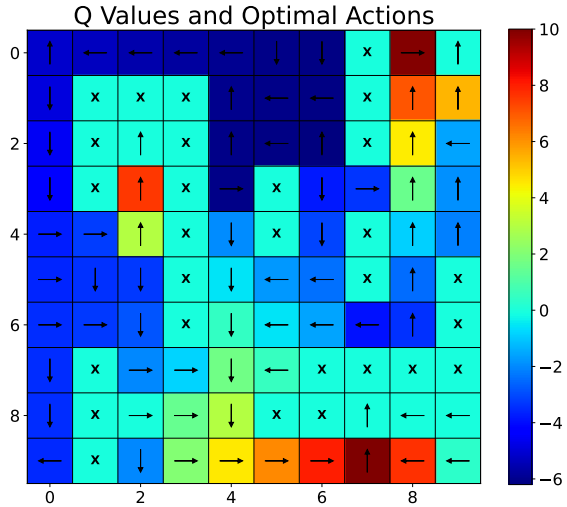
| | |
|---|---|
| Reward | -1 |
| Path taken | [36, 46, 56, 55, 54, 64, 74, 84, 94, 95, 96, 97, 87] |



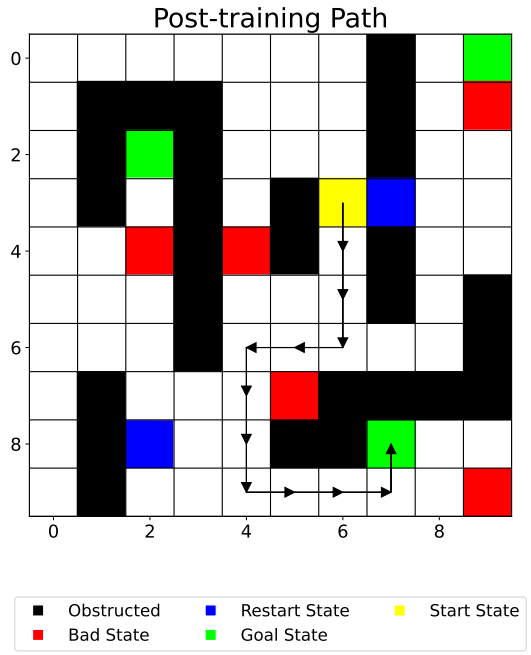



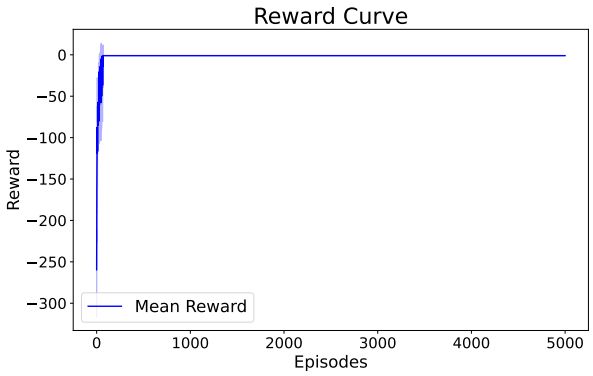



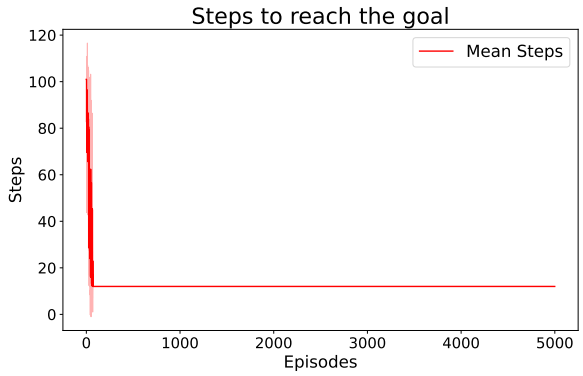
| Variables | Value |
|---|---|
| Wind parameter | `wind=False(clear)` |
| Transition Parameter ($p$) | 1.0 |
| Bias Parameter ($b$) | 0.5 |
| Starting Position | $(3, 6)$ |
| Algorithm | Q-Learning |
| Policy | Softmax |

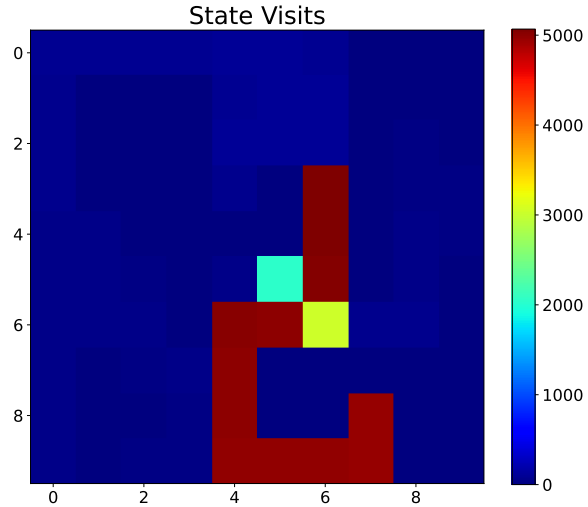
| Hyperparameter | Value |
|---|---|
| Learning rate ($\alpha$) | 0.30 |
| Discount factor ($\gamma$) | 0.99 |
| Temperature ($\tau$) | 0.01 |

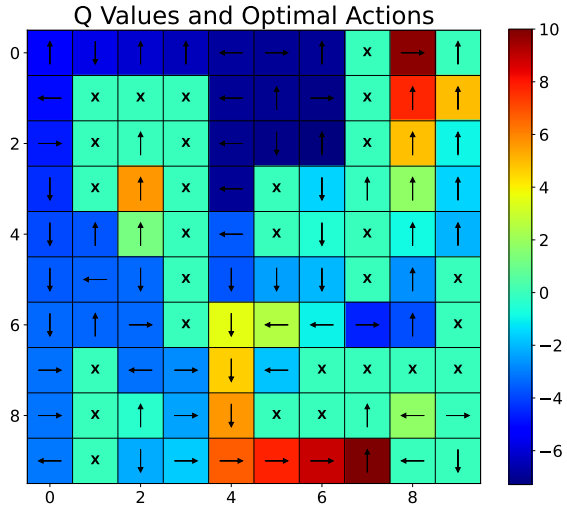
| | |
|---|---|
| Reward | -1 |
| Path taken | [36, 46, 56, 66, 65, 64, 74, 84, 94, 95, 96, 97, 87] |



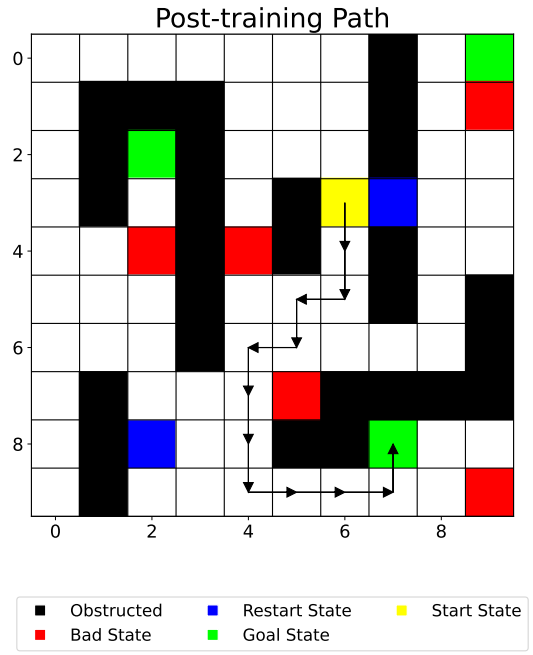



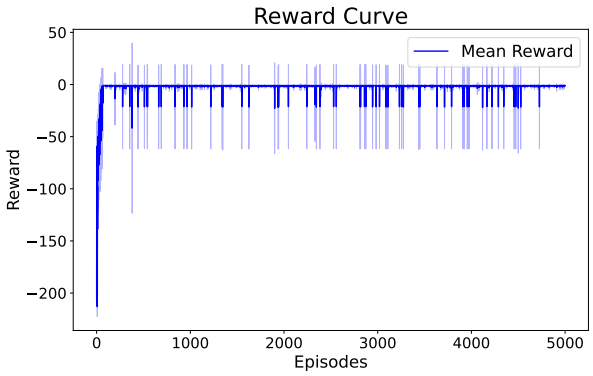



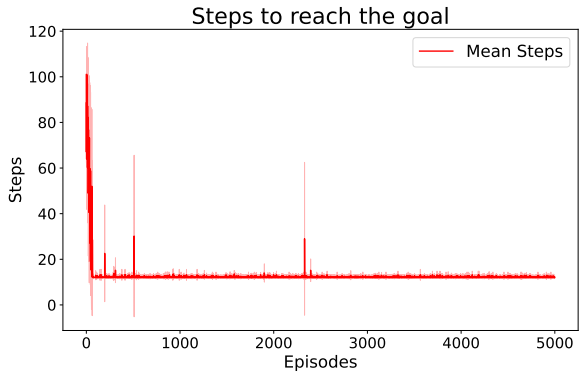
| Variables | Value |
|---|---|
| Wind parameter | `wind=False(clear)` |
| Transition Parameter ($p$) | 1.0 |
| Bias Parameter ($b$) | 0.5 |
| Starting Position | $(3, 6)$ |
| Algorithm | Q-Learning |
| Policy | $\epsilon-$greedy |

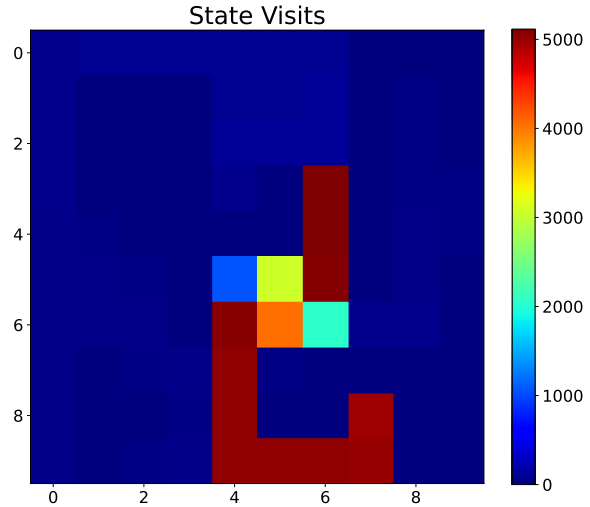
| Hyperparameter | Value |
|---|---|
| Learning rate ($\alpha$) | 0.30 |
| Discount factor ($\gamma$) | 0.99 |
| Temperature ($\tau$) | 0.01 |

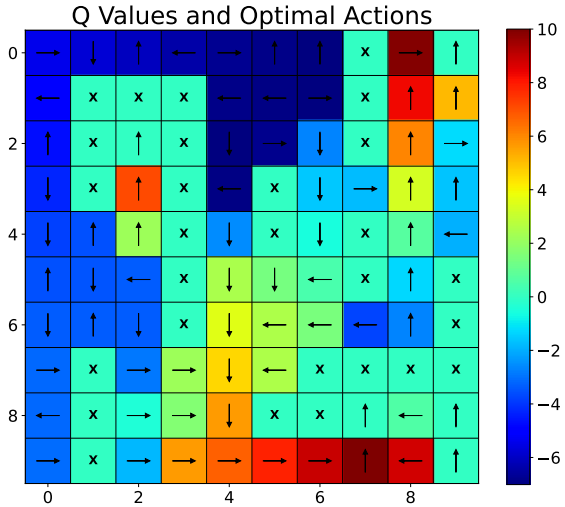
| | |
|---|---|
| Reward | -1 |
| Path taken | [36, 46, 56, 66, 65, 64, 74, 84, 94, 95, 96, 97, 87] |



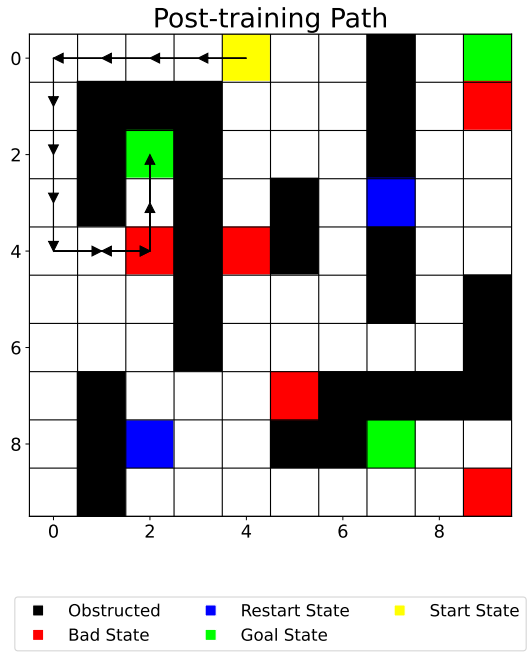



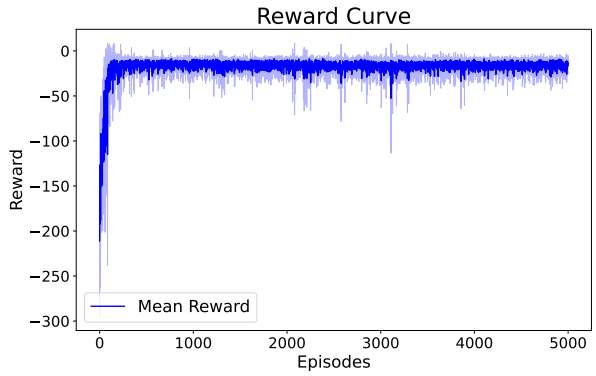



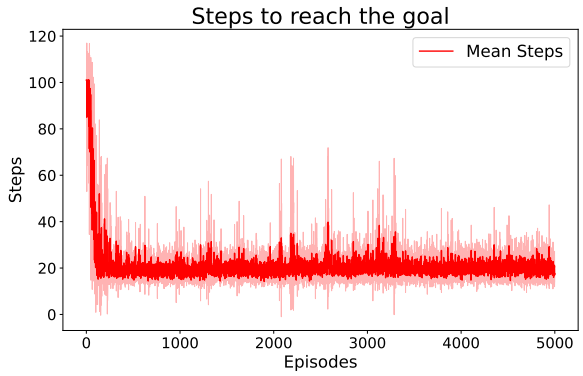
| Variables | Value |
|---|---|
| Wind parameter | `wind=False(clear)` |
| Transition Parameter ($p$) | 0.7 |
| Bias Parameter ($b$) | 0.5 |
| Starting Position | $(0, 4)$ |
| Algorithm | SARSA |
| Policy | Softmax |

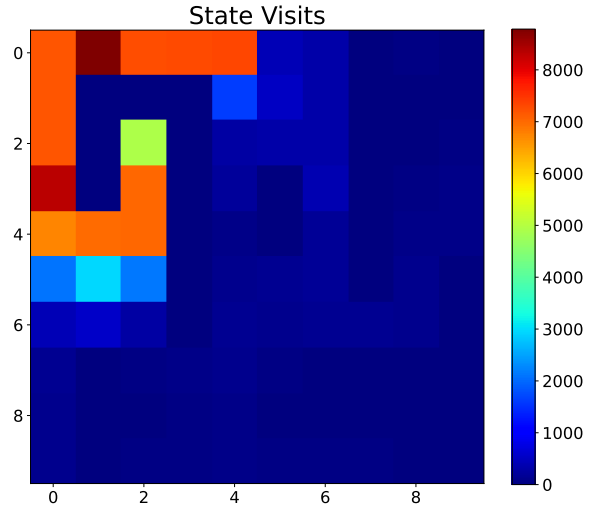
| Hyperparameter | Value |
|---|---|
| Learning rate ($\alpha$) | 0.30 |
| Discount factor ($\gamma$) | 0.99 |
| Temperature ($\tau$) | 0.05 |

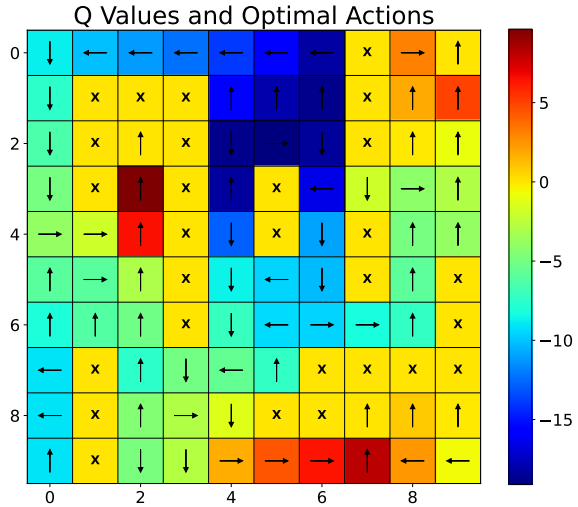
| | |
|---|---|
| Reward | -17 |
| Path taken | [4, 3, 3, 3, 2, 1, 0, 10, 20, 30, 40, 41, 42, 41, 41, 42, 32, 32, 22] |



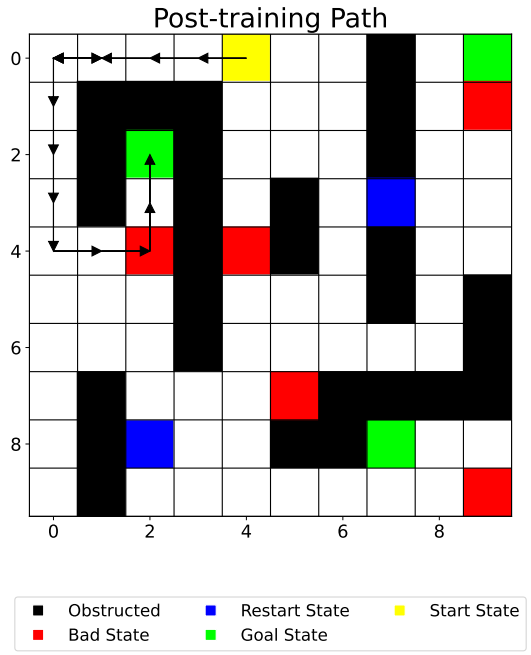



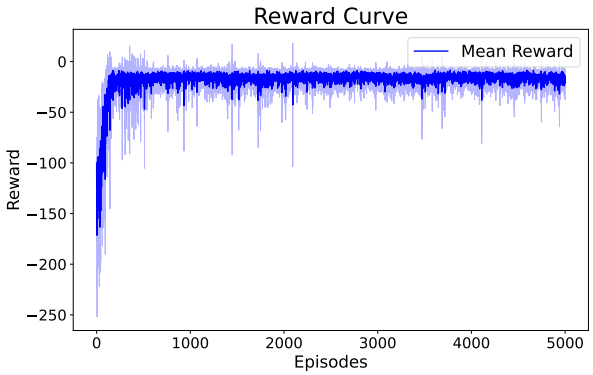



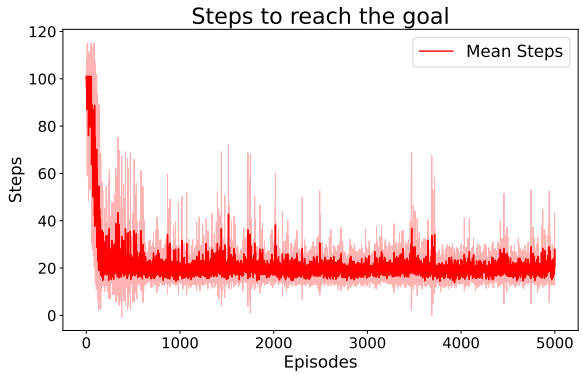
| Variables | Value |
|---|---|
| Wind parameter | `wind=False(clear)` |
| Transition Parameter ($p$) | 0.7 |
| Bias Parameter ($b$) | 0.5 |
| Starting Position | $(0, 4)$ |
| Algorithm | SARSA |
| Policy | $\epsilon-$greedy |

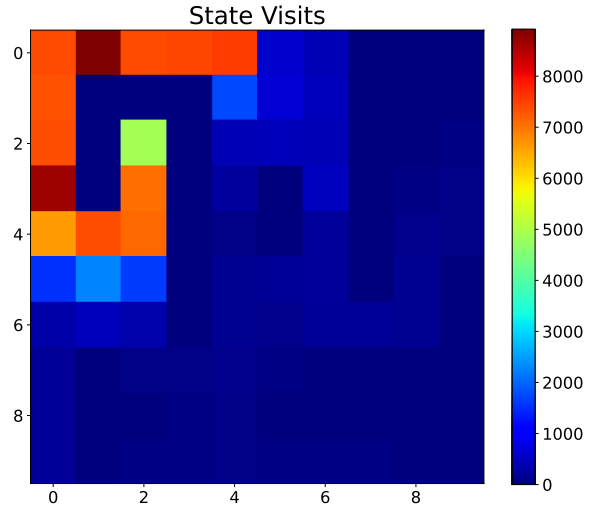
| Hyperparameter | Value |
|---|---|
| Learning rate ($\alpha$) | 0.20 |
| Discount factor ($\gamma$) | 0.95 |
| Epsilon ($\epsilon$) | 0.01 |

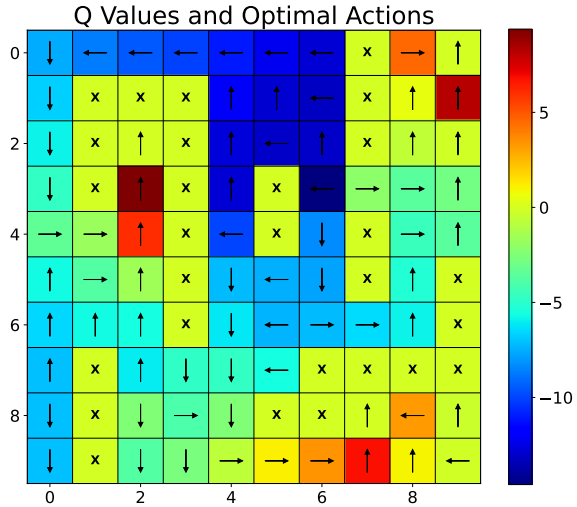
| | |
|---|---|
| Reward | -12 |
| Path taken | [4, 4, 3, 2, 1, 0, 1, 0, 1, 1, 0, 10, 20, 30, 40, 41, 42, 32, 22] |



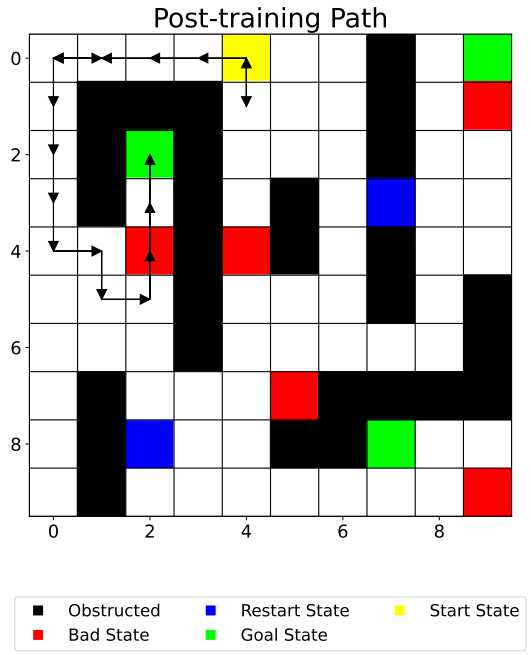



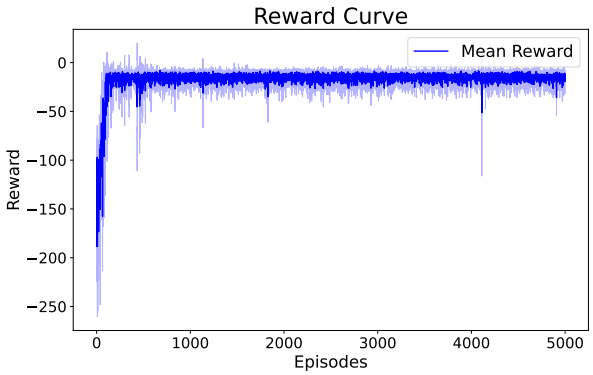



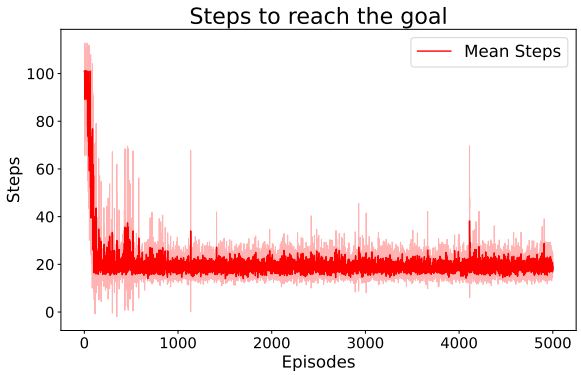
| Variables | Value |
|---|---|
| Wind parameter | `wind=False(clear)` |
| Transition Parameter ($p$) | 0.7 |
| Bias Parameter ($b$) | 0.5 |
| Starting Position | $(0, 4)$ |
| Algorithm | Q-Learning |
| Policy | Softmax |

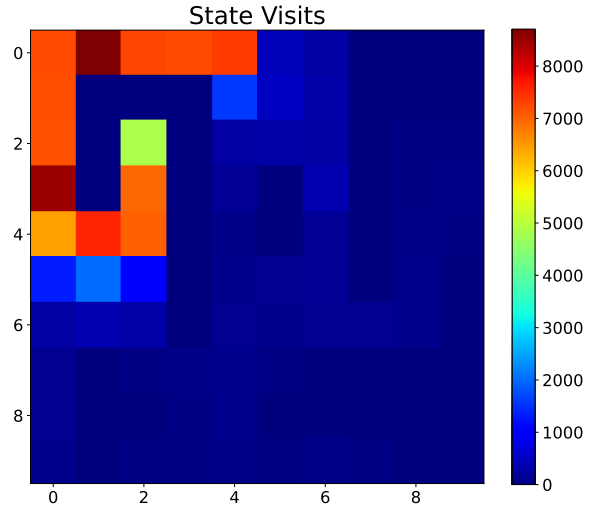
| Hyperparameter | Value |
|---|---|
| Learning rate ($\alpha$) | 0.30 |
| Discount factor ($\gamma$) | 0.99 |
| Temperature ($\tau$) | 0.10 |

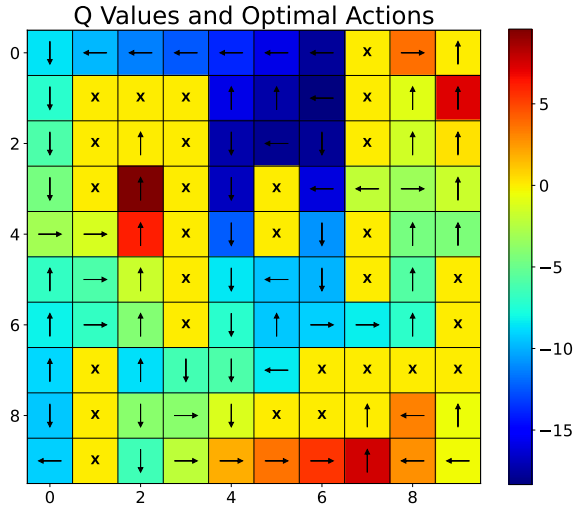
| | |
|---|---|
| Reward | -17 |
| Path taken | [4, 14, 4, 3, 3, 2, 1, 0, 0, 1, 0, 10, 10, 10, 10, 20, 30, 40, 41, 51, 52, 42, 32, 22] |



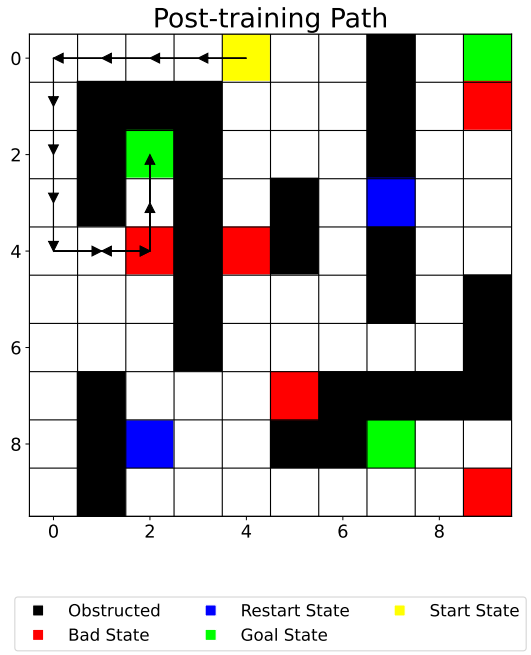



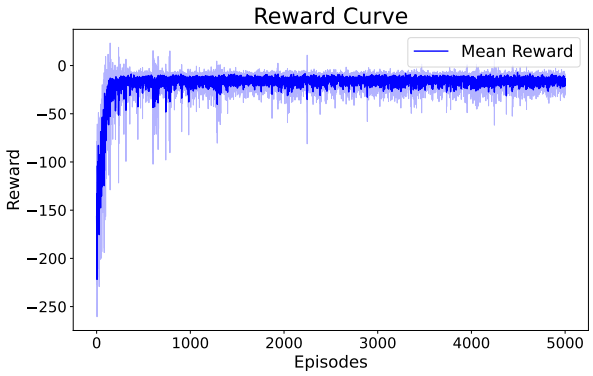



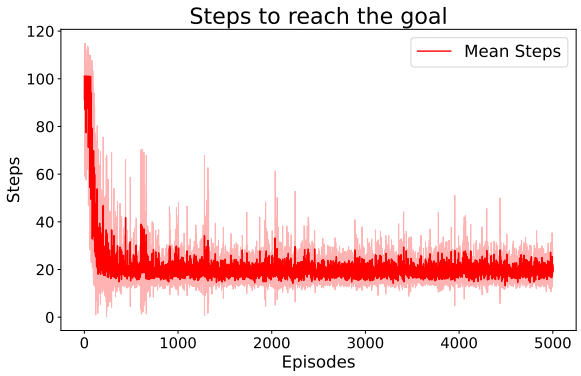
| Variables | Value |
|---|---|
| Wind parameter | `wind=False(clear)` |
| Transition Parameter ($p$) | 0.7 |
| Bias Parameter ($b$) | 0.5 |
| Starting Position | $(0, 4)$ |
| Algorithm | Q-Learning |
| Policy | $\epsilon-$greedy |

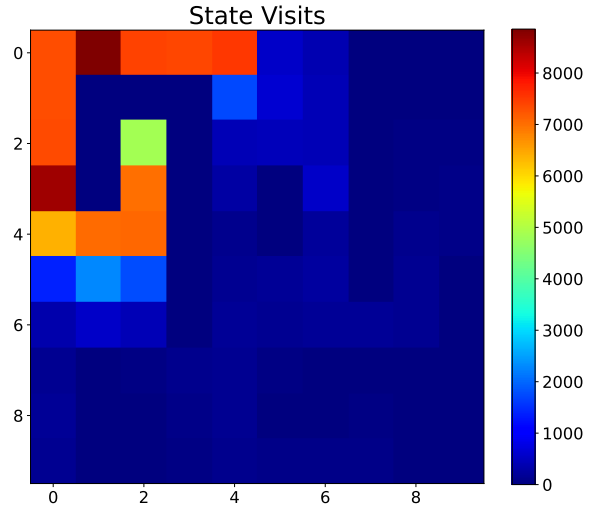
| Hyperparameter | Value |
|---|---|
| Learning rate ($\alpha$) | 0.20 |
| Discount factor ($\gamma$) | 0.99 |
| Epsilon ($\epsilon$) | 0.01 |

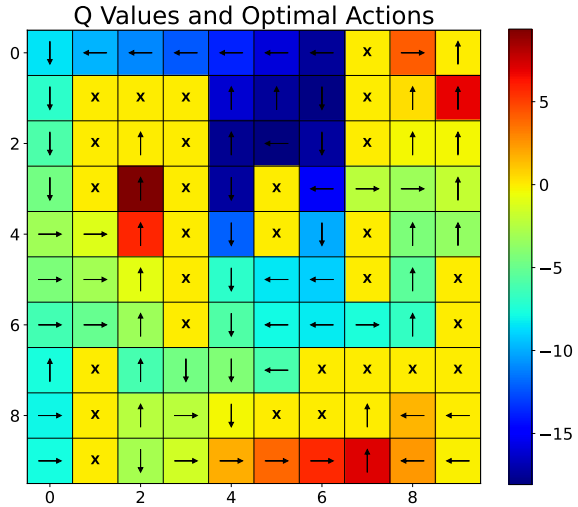
| | |
|---|---|
| Reward | -14 |
| Path taken | [4, 3, 2, 1, 0, 10, 20, 30, 40, 41, 42, 41, 42, 32, 32, 22] |



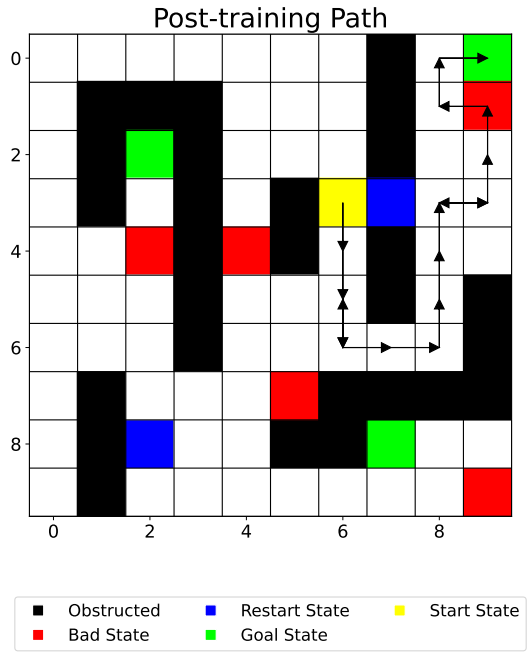



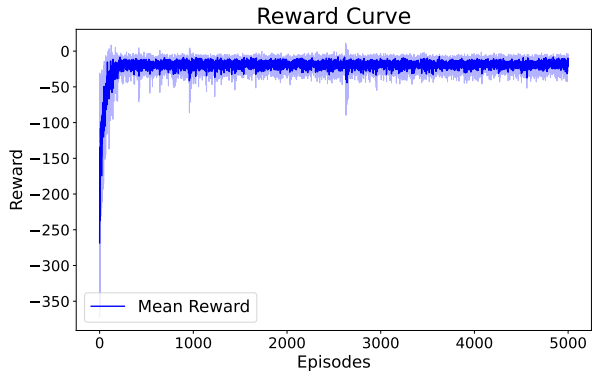



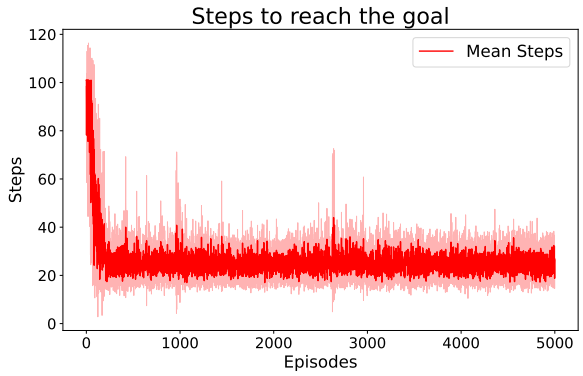
| Variables | Value |
|---|---|
| Wind parameter | `wind=False(clear)` |
| Transition Parameter ($p$) | 0.7 |
| Bias Parameter ($b$) | 0.5 |
| Starting Position | $(3, 6)$ |
| Algorithm | SARSA |
| Policy | Softmax |

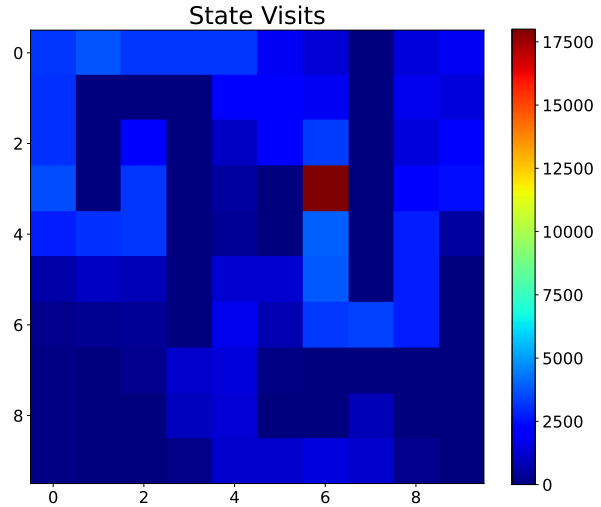
| Hyperparameter | Value |
|---|---|
| Learning rate ($\alpha$) | 0.30 |
| Discount factor ($\gamma$) | 0.99 |
| Temperature ($\tau$) | 0.01 |

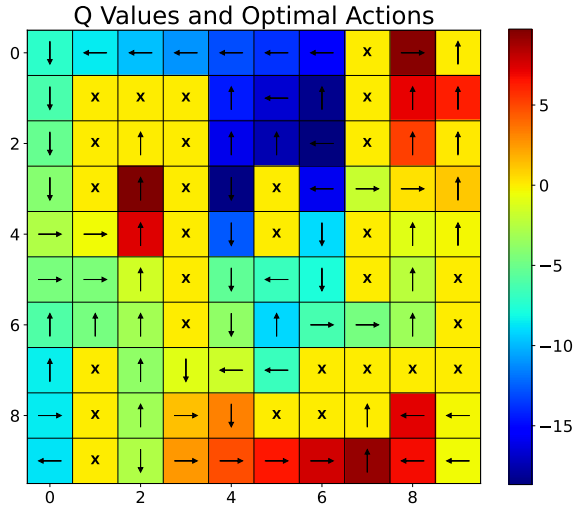
| | |
|---|---|
| Reward | -13 |
| Path taken | [36, 46, 56, 66, 56, 66, 66, 67, 68, 58, 48, 38, 39, 38, 39, 29, 19, 18, 8, 9] |



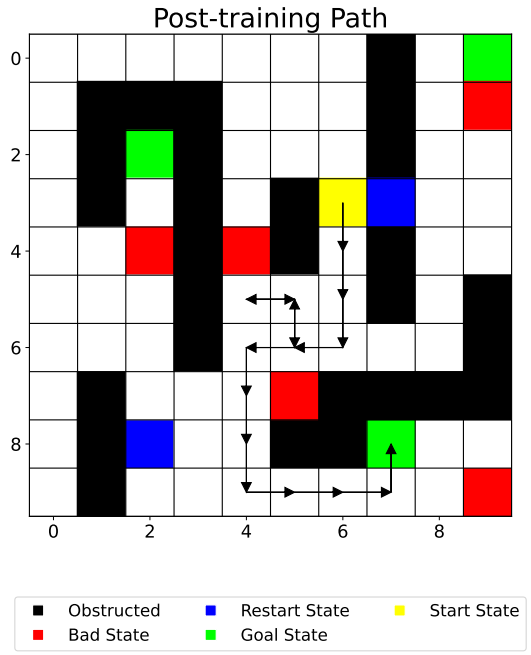



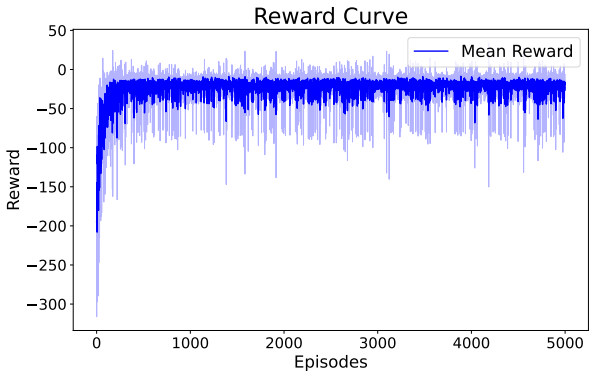



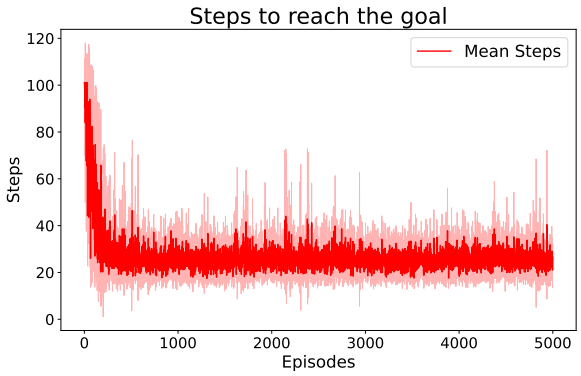
| Variables | Value |
|---|---|
| Wind parameter | `wind=False(clear)` |
| Transition Parameter ($p$) | 0.7 |
| Bias Parameter ($b$) | 0.5 |
| Starting Position | $(3, 6)$ |
| Algorithm | SARSA |
| Policy | $\epsilon-$greedy |

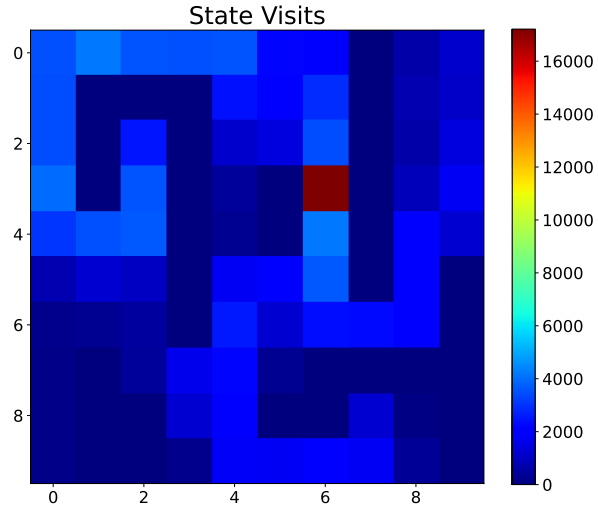
| Hyperparameter | Value |
|---|---|
| Learning rate ($\alpha$) | 0.20 |
| Discount factor ($\gamma$) | 0.99 |
| Epsilon ($\epsilon$) | 0.01 |

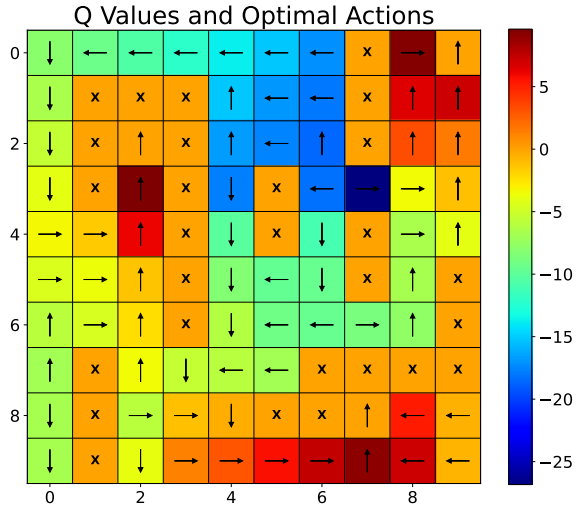
| | |
|---|---|
| Reward | -7 |
| Path taken | [36, 36, 36, 46, 56, 66, 65, 55, 54, 55, 65, 64, 74, 84, 94, 95, 96, 97, 87] |



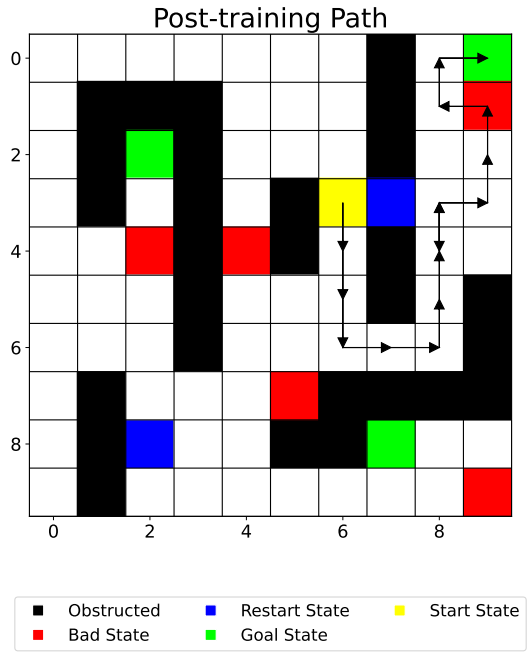



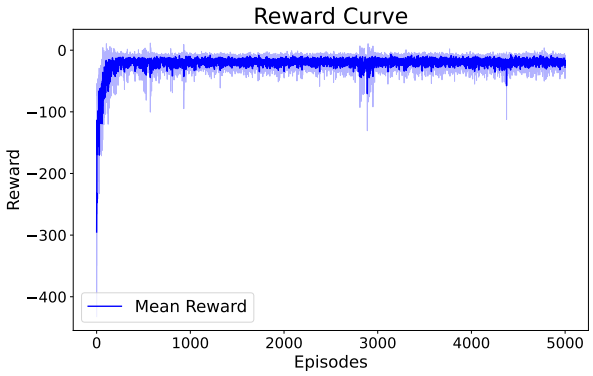



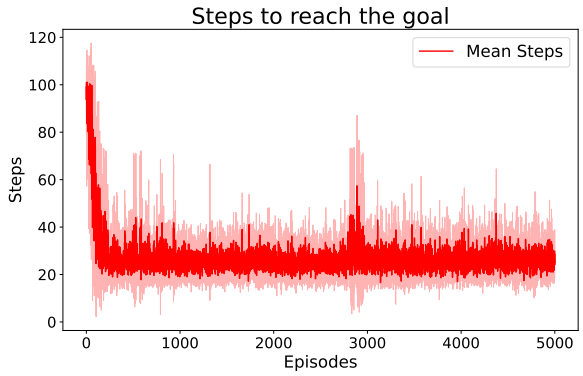
| Variables | Value |
|---|---|
| Wind parameter | `wind=False(clear)` |
| Transition Parameter ($p$) | 0.7 |
| Bias Parameter ($b$) | 0.5 |
| Starting Position | $(3, 6)$ |
| Algorithm | Q-Learning |
| Policy | Softmax |

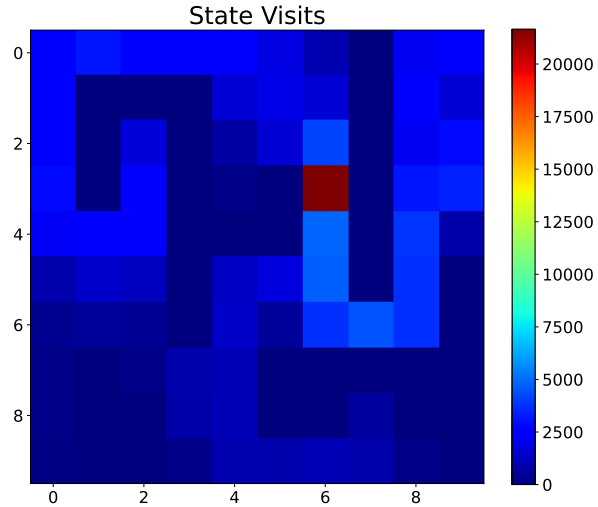
| Hyperparameter | Value |
|---|---|
| Learning rate ($\alpha$) | 0.30 |
| Discount factor ($\gamma$) | 0.95 |
| Temperature ($\tau$) | 0.01 |

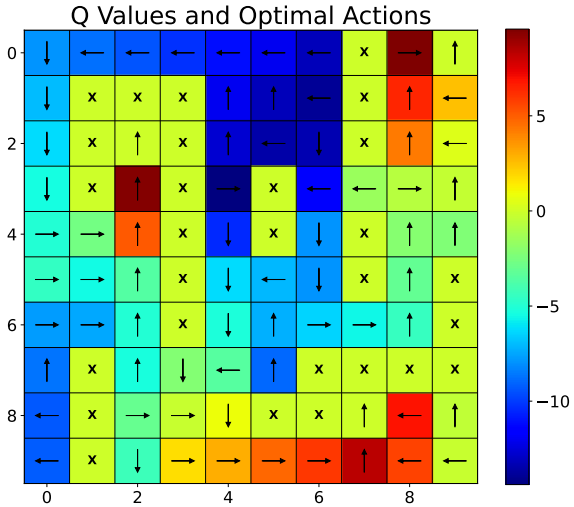
| | |
|---|---|
| Reward | -16 |
| Path taken | [36, 36, 36, 36, 46, 56, 66, 67, 67, 68, 58, 48, 48, 38, 48, 38, 39, 39, 29, 19, 18, 8, 9] |



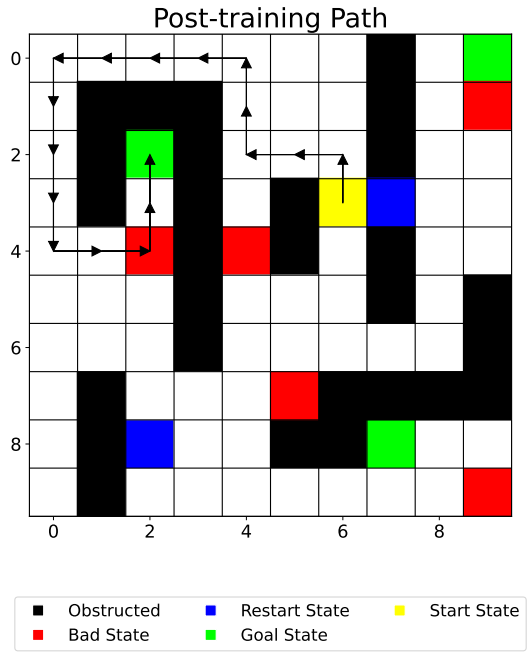



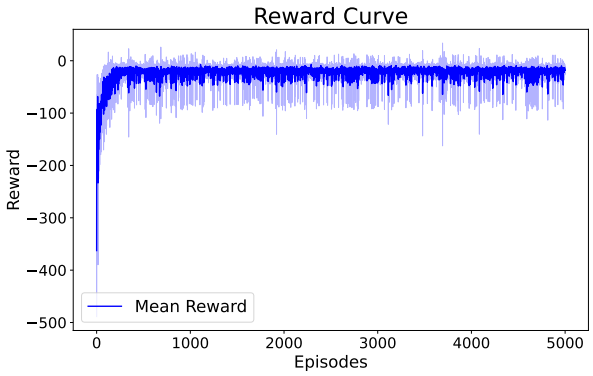



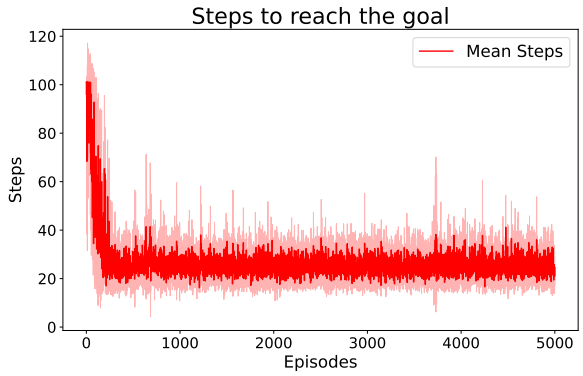
| Variables | Value |
|---|---|
| Wind parameter | `wind=False(clear)` |
| Transition Parameter ($p$) | 0.7 |
| Bias Parameter ($b$) | 0.5 |
| Starting Position | $(3, 6)$ |
| Algorithm | Q-Learning |
| Policy | $\epsilon-$greedy |

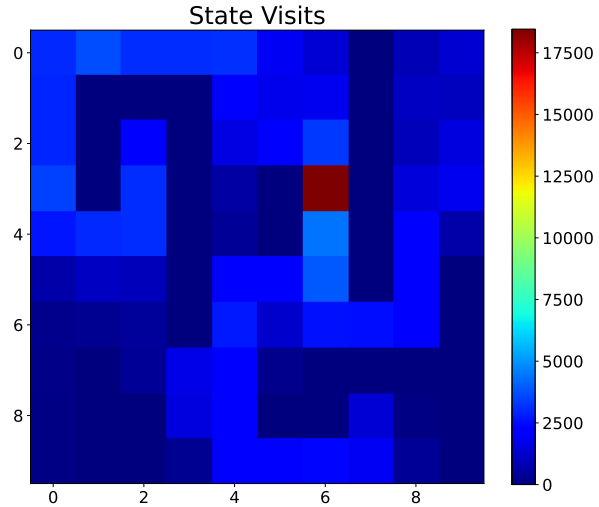
| Hyperparameter | Value |
|---|---|
| Learning rate ($\alpha$) | 0.20 |
| Discount factor ($\gamma$) | 0.99 |
| Epsilon ($\epsilon$) | 0.01 |

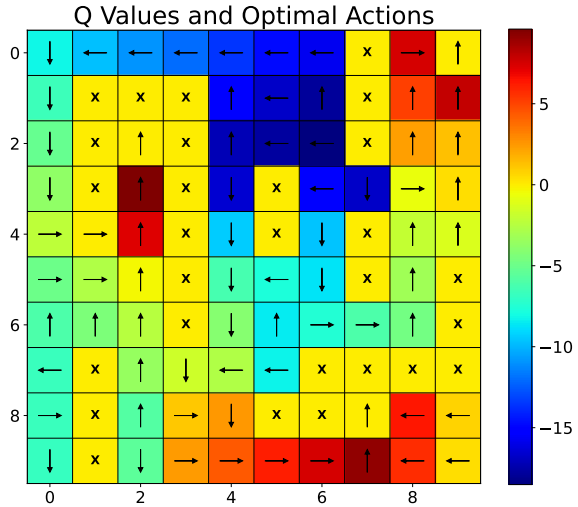
| | |
|---|---|
| Reward | -24 |
| Path taken | [36, 36, 36, 26, 25, 24, 24, 24, 14, 14, 4, 3, 2, 1, 0, 0, 0, 10, 20, 30, 40, 41, 42, 42, 32, 22] |



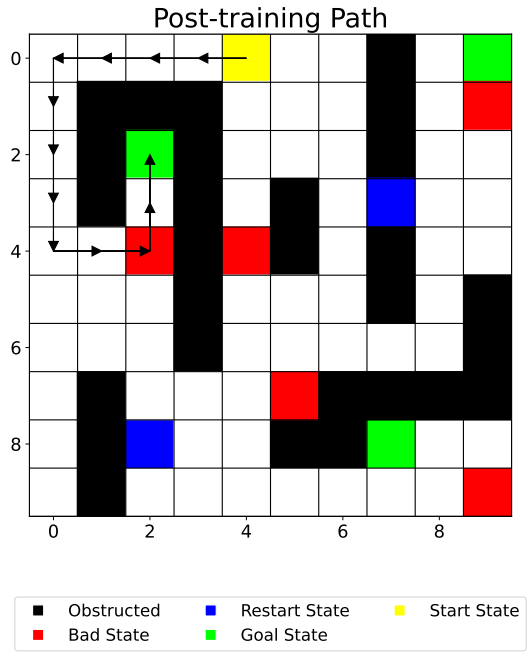



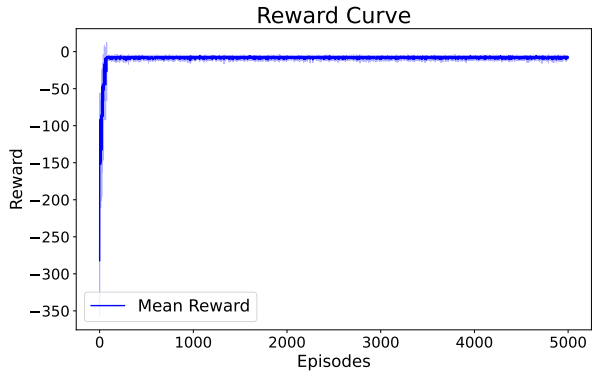



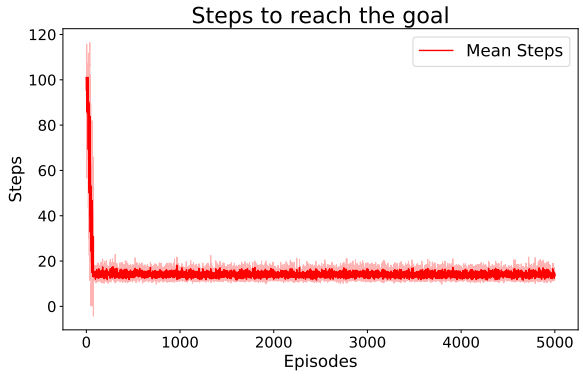
| Variables | Value |
|---|---|
| Wind parameter | `wind=True(windy)` |
| Transition Parameter ($p$) | 1.0 |
| Bias Parameter ($b$) | 0.5 |
| Starting Position | $(0, 4)$ |
| Algorithm | SARSA |
| Policy | Softmax |

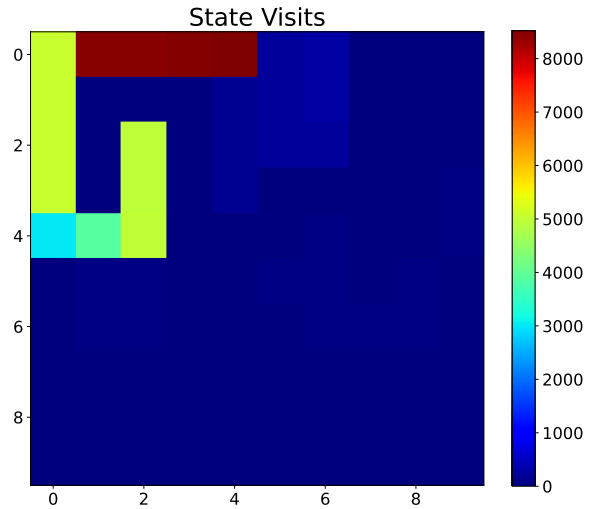
| Hyperparameter | Value |
|---|---|
| Learning rate ($\alpha$) | 0.30 |
| Discount factor ($\gamma$) | 0.90 |
| Temperature ($\tau$) | 0.01 |

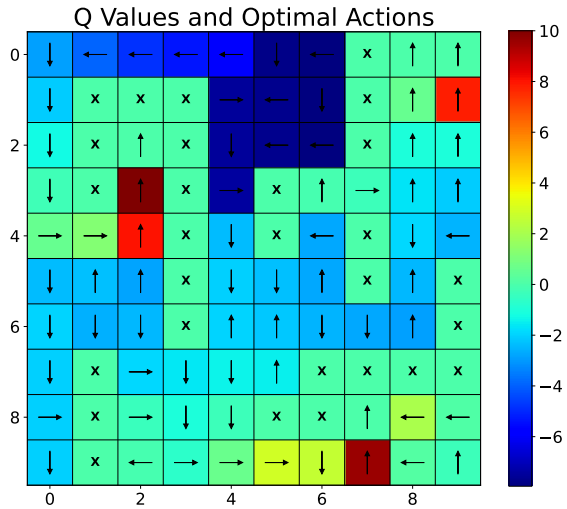
| | |
|---|---|
| Reward | -8 |
| Path taken | [4, 3, 2, 1, 1, 1, 0, 10, 20, 30, 40, 41, 42, 32, 22] |



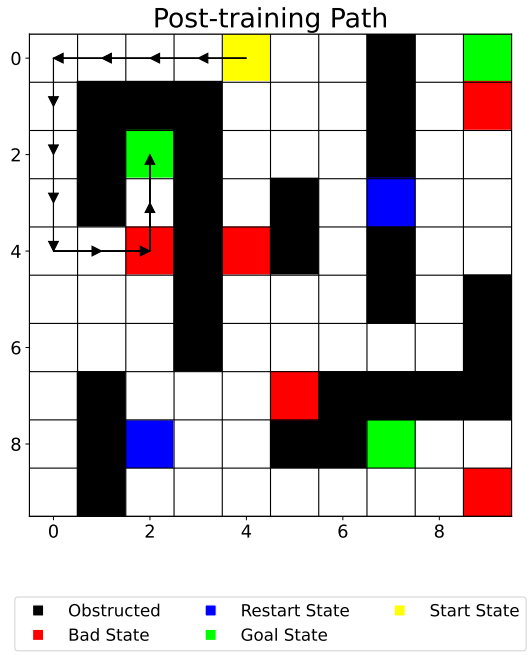



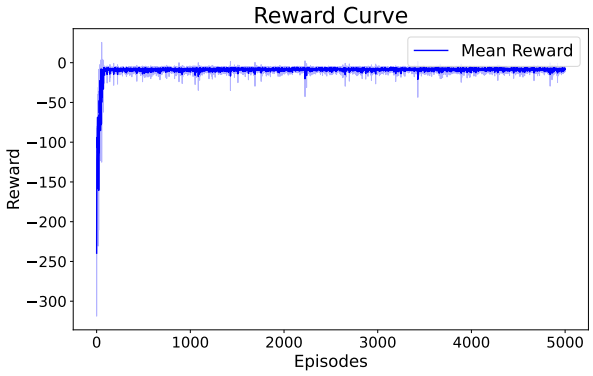



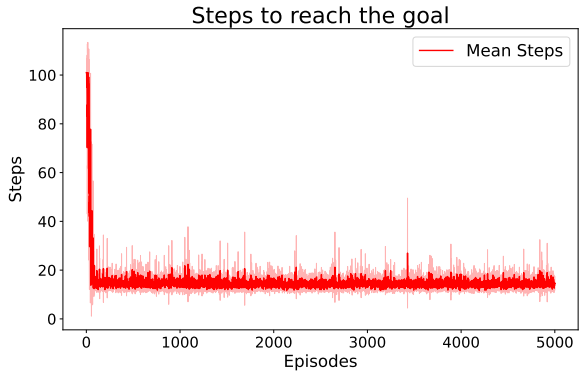
| Variables | Value |
|---|---|
| Wind parameter | `wind=True(windy)` |
| Transition Parameter ($p$) | 1.0 |
| Bias Parameter ($b$) | 0.5 |
| Starting Position | $(0, 4)$ |
| Algorithm | SARSA |
| Policy | $\epsilon-$greedy |

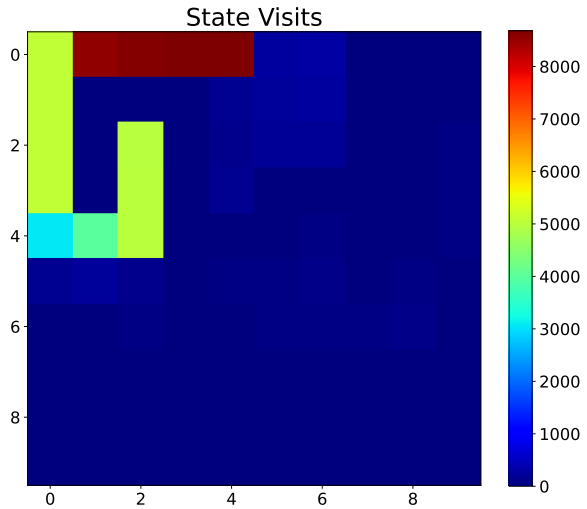
| Hyperparameter | Value |
|---|---|
| Learning rate ($\alpha$) | 0.30 |
| Discount factor ($\gamma$) | 0.95 |
| Epsilon ($\epsilon$) | 0.01 |

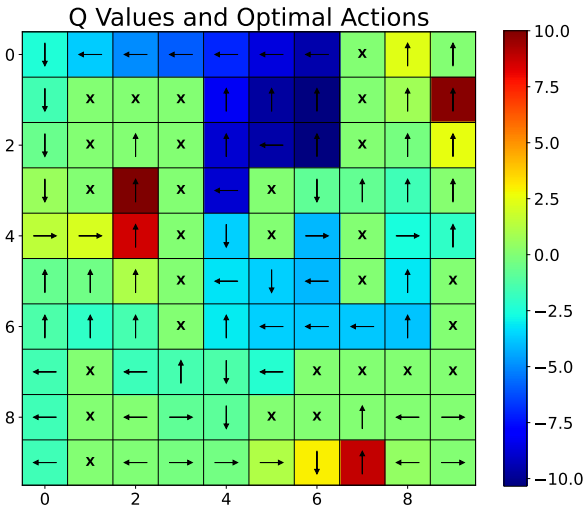
| | |
|---|---|
| Reward | -10 |
| Path taken | [4, 4, 3, 3, 2, 2, 2, 1, 0, 10, 20, 30, 40, 41, 42, 32, 22] |



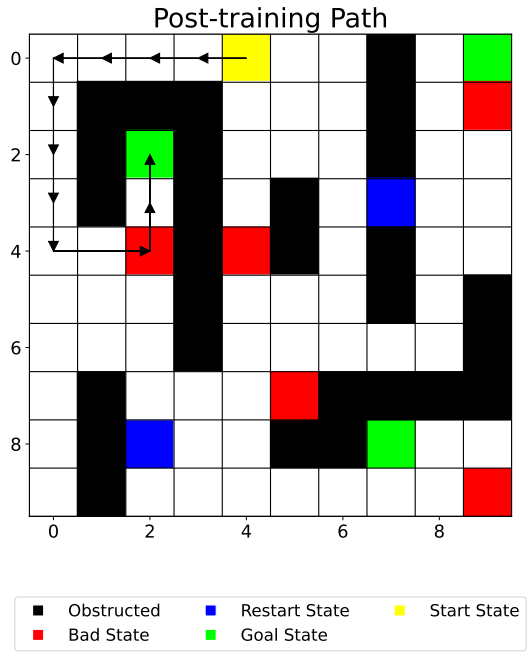



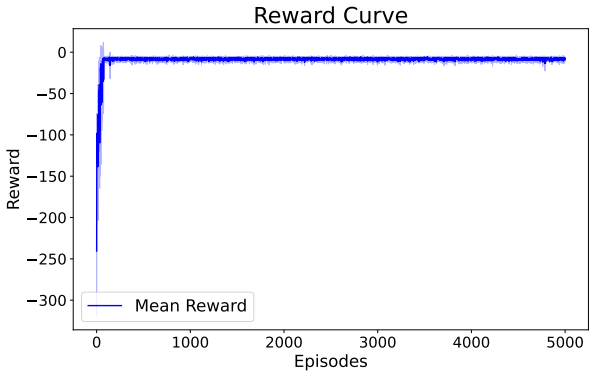



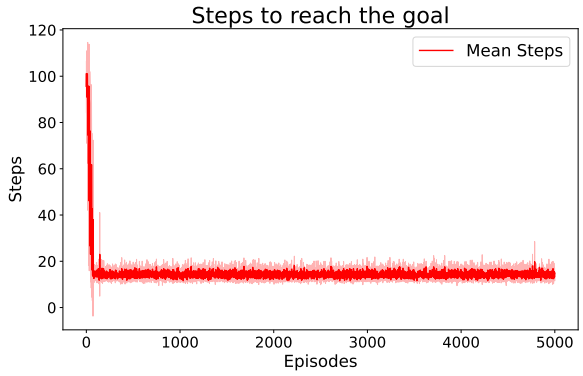
| Variables | Value |
|---|---|
| Wind parameter | `wind=True(windy)` |
| Transition Parameter ($p$) | 1.0 |
| Bias Parameter ($b$) | 0.5 |
| Starting Position | $(0, 4)$ |
| Algorithm | Q-Learning |
| Policy | Softmax |

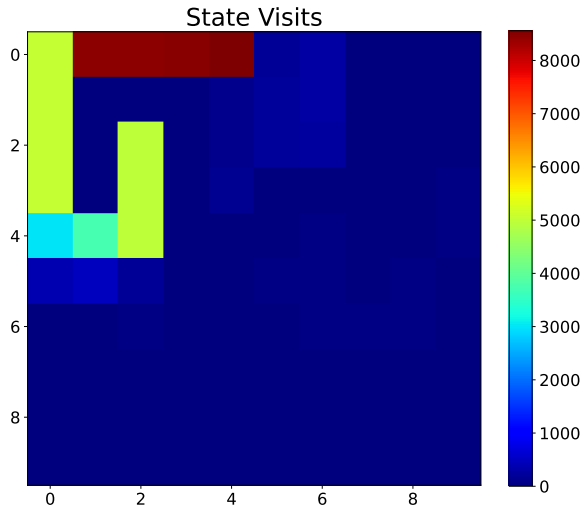
| Hyperparameter | Value |
|---|---|
| Learning rate ($\alpha$) | 0.30 |
| Discount factor ($\gamma$) | 0.90 |
| Temperature ($\tau$) | 0.01 |

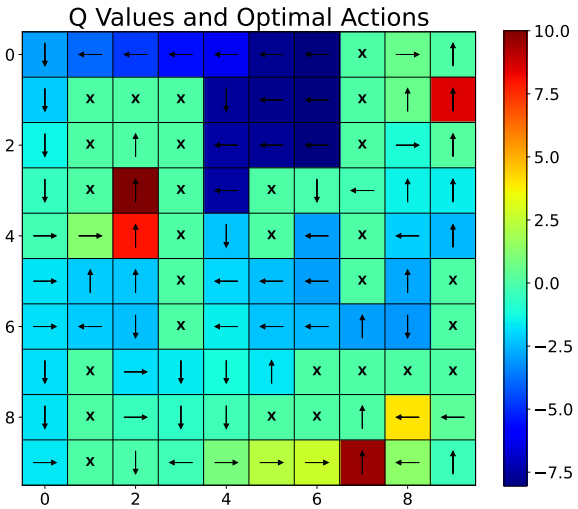
| | |
|---|---|
| Reward | -7 |
| Path taken | [4, 3, 2, 2, 2, 1, 0, 10, 20, 30, 40, 42, 32, 22] |



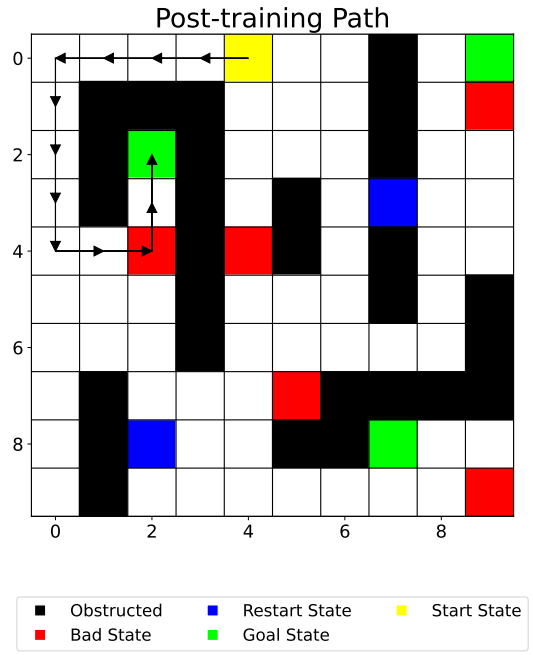



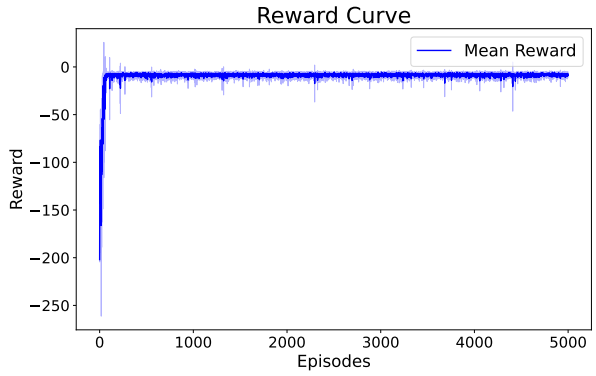



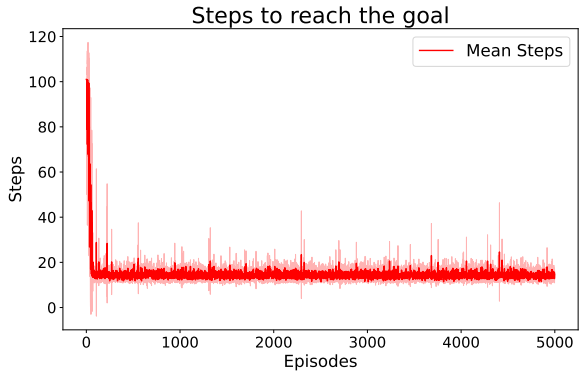
| Variables | Value |
|---|---|
| Wind parameter | `wind=True(windy)` |
| Transition Parameter ($p$) | 1.0 |
| Bias Parameter ($b$) | 0.5 |
| Starting Position | $(0, 4)$ |
| Algorithm | Q-Learning |
| Policy | $\epsilon-$greedy |

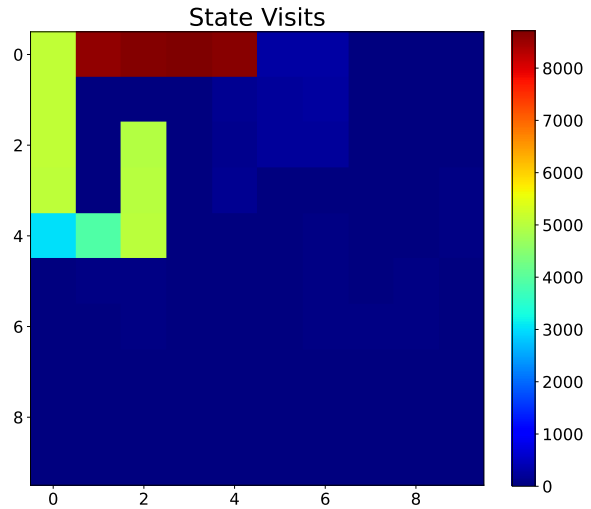
| Hyperparameter | Value |
|---|---|
| Learning rate ($\alpha$) | 0.30 |
| Discount factor ($\gamma$) | 0.90 |
| Epsilon ($\epsilon$) | 0.01 |

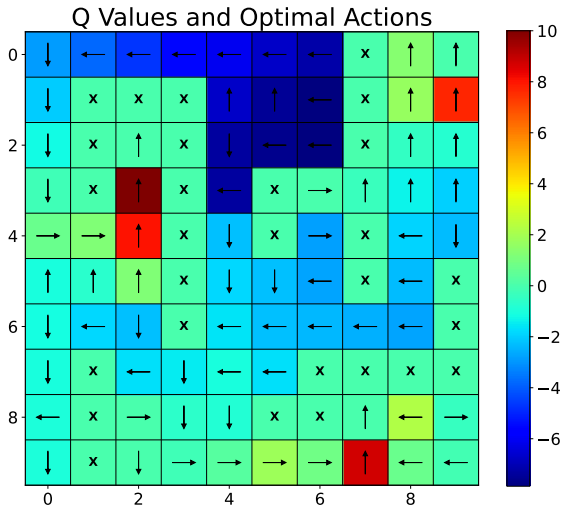
| | |
|---|---|
| Reward | -6 |
| Path taken | [4, 3, 2, 1, 0, 10, 20, 30, 40, 41, 42, 32, 22] |



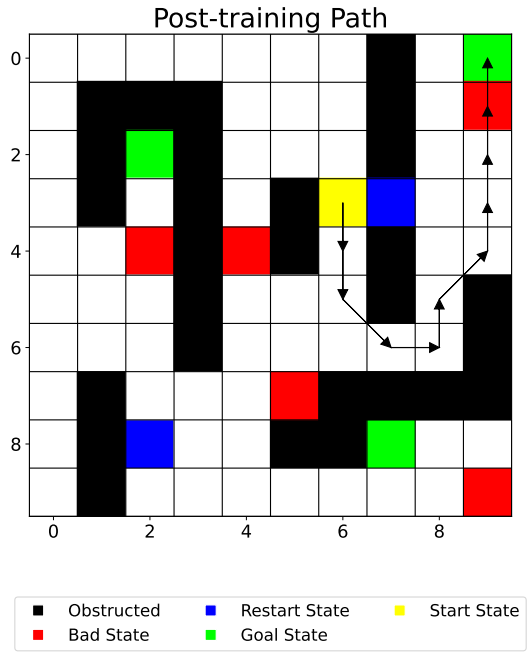



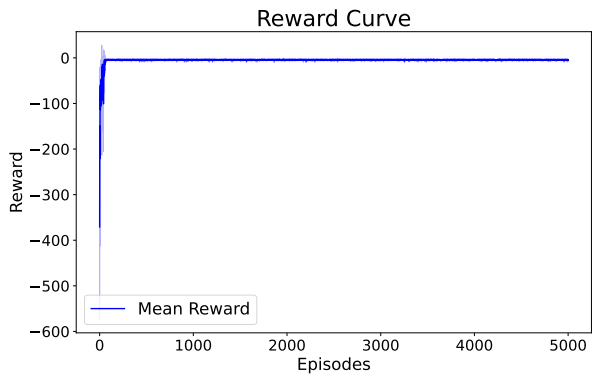



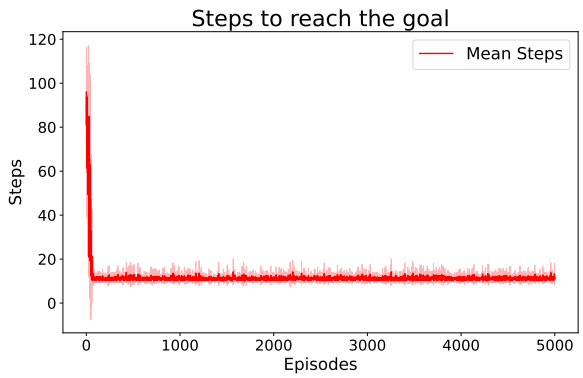
| Variables | Value |
|---|---|
| Wind parameter | `wind=True(windy)` |
| Transition Parameter ($p$) | 1.0 |
| Bias Parameter ($b$) | 0.5 |
| Starting Position | $(3, 6)$ |
| Algorithm | SARSA |
| Policy | Softmax |

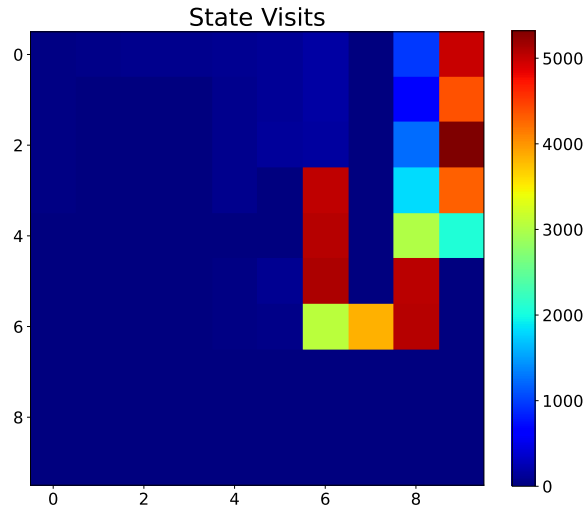
| Hyperparameter | Value |
|---|---|
| Learning rate ($\alpha$) | 0.30 |
| Discount factor ($\gamma$) | 0.99 |
| Temperature ($\tau$) | 0.05 |

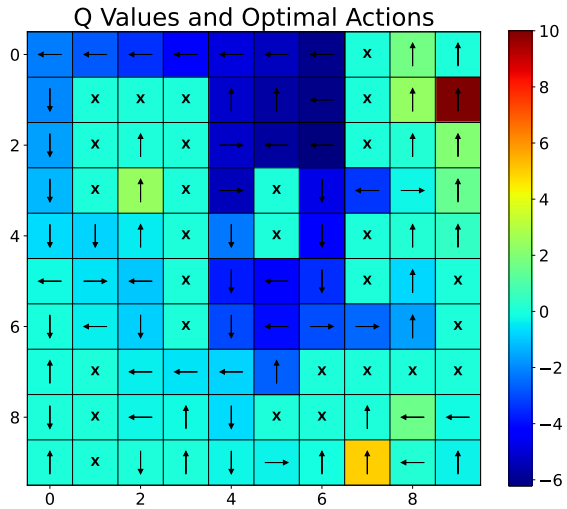
| | |
|---|---|
| Reward | -4 |
| Path taken | [36, 46, 56, 67, 68, 58, 49, 39, 29, 19, 9] |



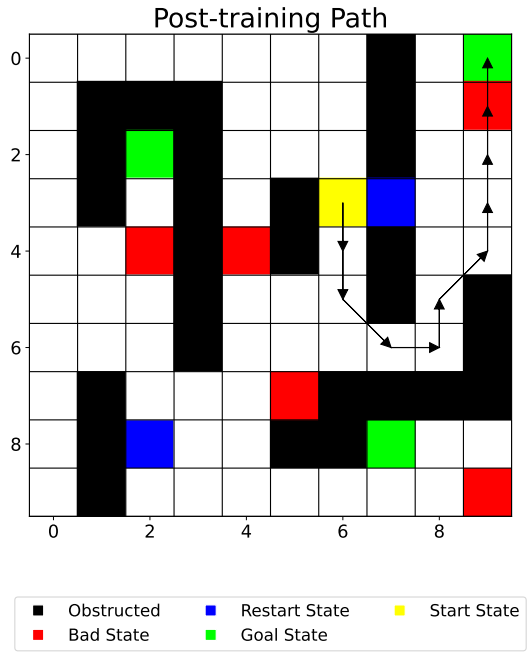



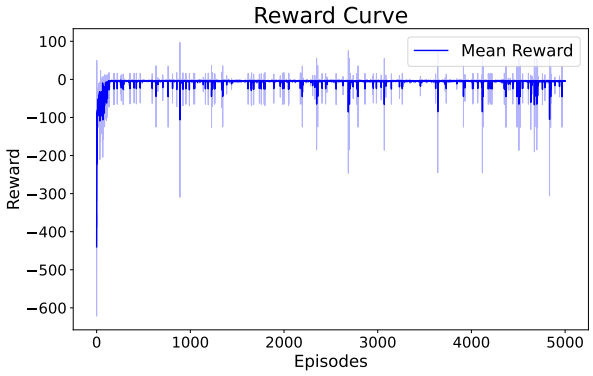



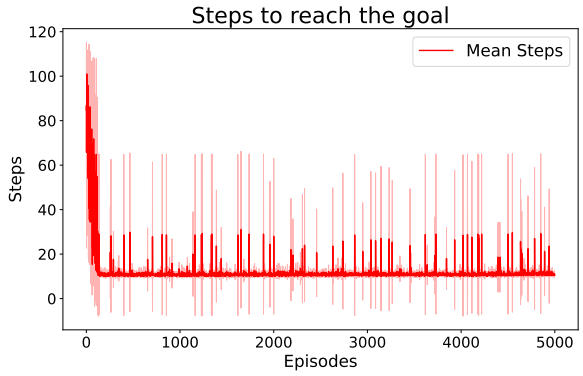
| Variables | Value |
|---|---|
| Wind parameter | `wind=True(windy)` |
| Transition Parameter ($p$) | 1.0 |
| Bias Parameter ($b$) | 0.5 |
| Starting Position | $(3, 6)$ |
| Algorithm | SARSA |
| Policy | $\epsilon-$greedy |

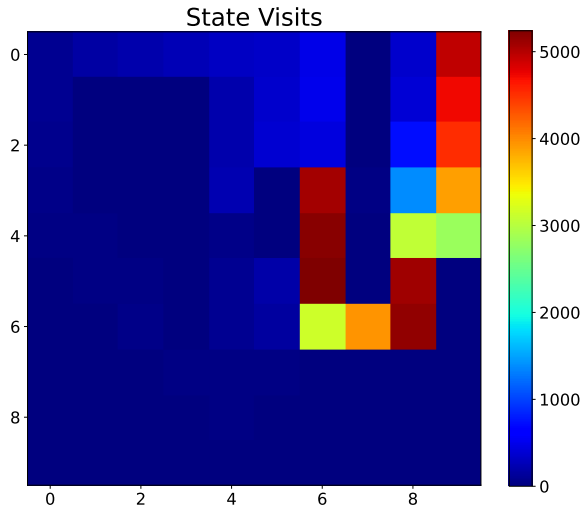
| Hyperparameter | Value |
|---|---|
| Learning rate ($\alpha$) | 0.10 |
| Discount factor ($\gamma$) | 0.99 |
| Temperature ($\tau$) | 0.01 |

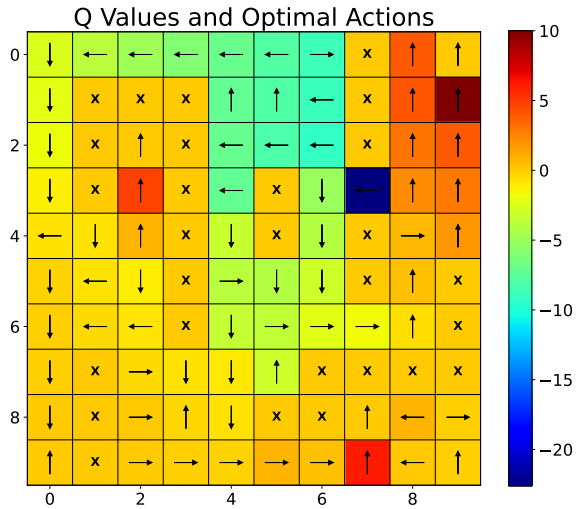
| | |
|---|---|
| Reward | -4 |
| Path taken | [36, 46, 56, 67, 68, 58, 49, 39, 29, 19, 9] |



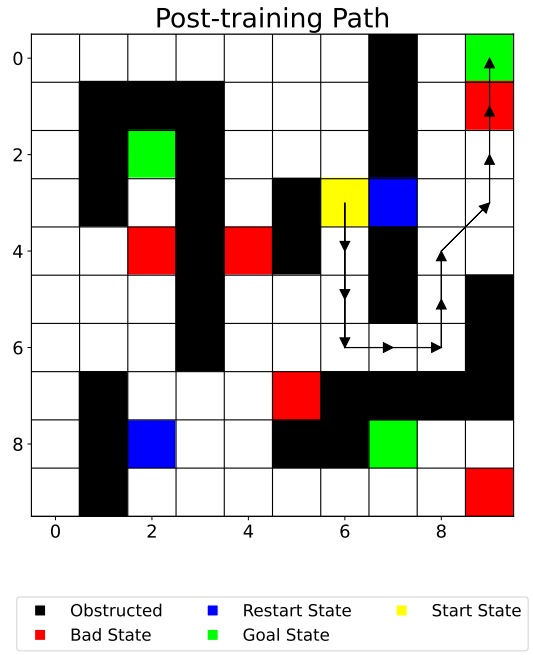



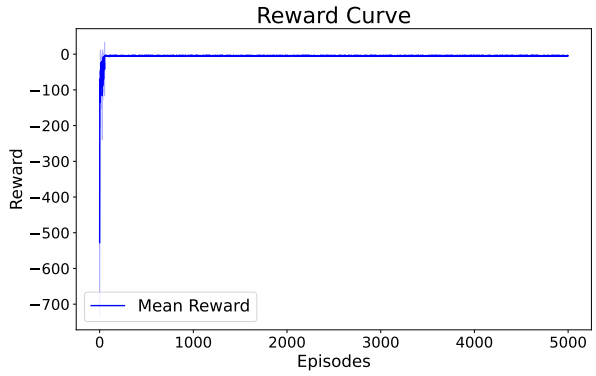



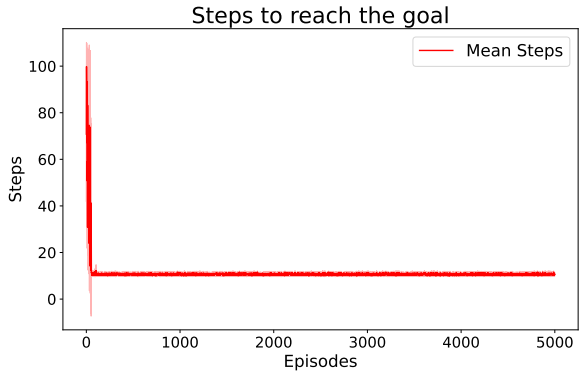
| Variables | Value |
|---|---|
| Wind parameter | `wind=True(windy)` |
| Transition Parameter ($p$) | 1.0 |
| Bias Parameter ($b$) | 0.5 |
| Starting Position | $(3, 6)$ |
| Algorithm | Q-Learning |
| Policy | Softmax |

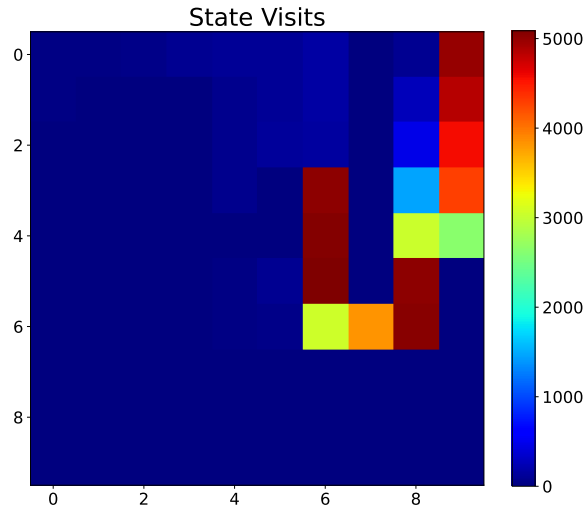
| Hyperparameter | Value |
|---|---|
| Learning rate ($\alpha$) | 0.30 |
| Discount factor ($\gamma$) | 0.95 |
| Temperature ($\tau$) | 0.10 |

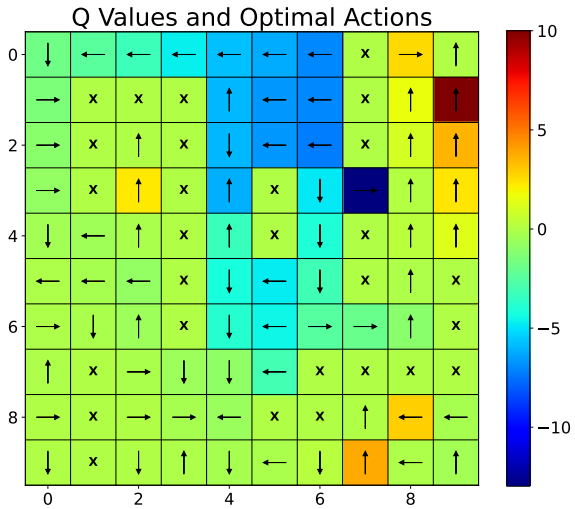
| | |
|---|---|
| Reward | -5 |
| Path taken | [36, 46, 56, 66, 67, 68, 58, 48, 39, 29, 19, 9] |



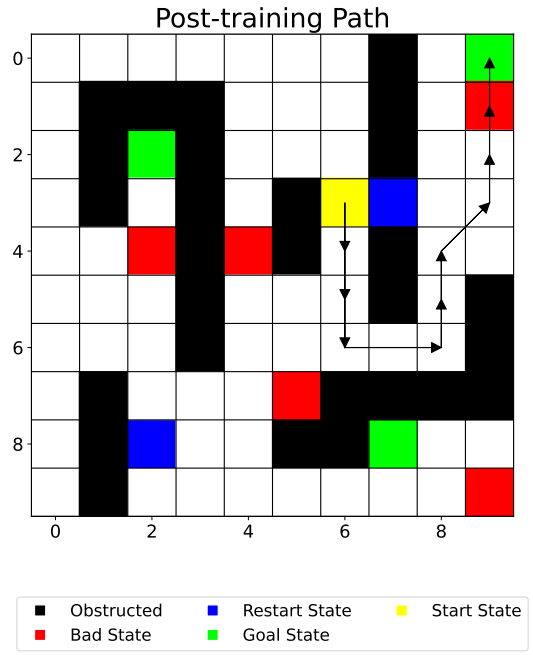



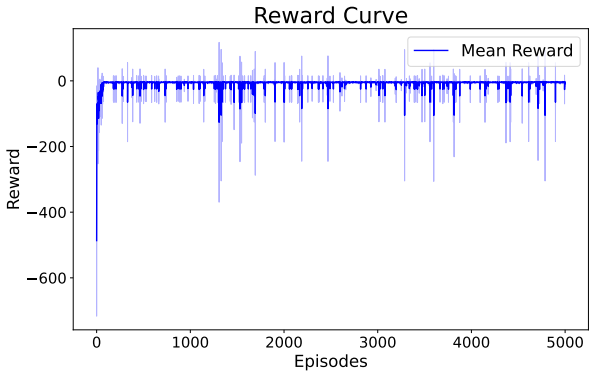



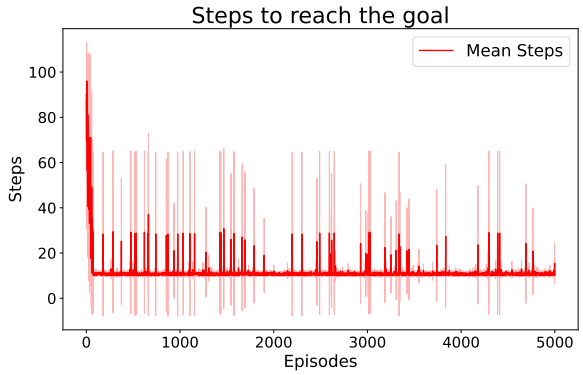
| Variables | Value |
|---|---|
| Wind parameter | `wind=True(windy)` |
| Transition Parameter ($p$) | 1.0 |
| Bias Parameter ($b$) | 0.5 |
| Starting Position | $(3, 6)$ |
| Algorithm | Q-Learning |
| Policy | $\epsilon-$greedy |

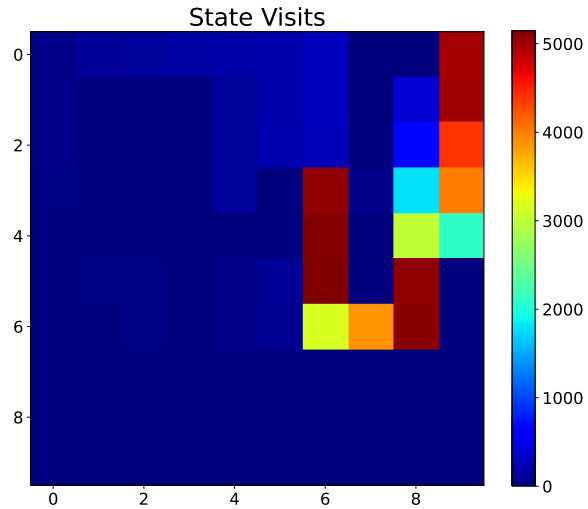
| Hyperparameter | Value |
|---|---|
| Learning rate ($\alpha$) | 0.20 |
| Discount factor ($\gamma$) | 0.90 |
| Temperature ($\tau$) | 0.01 |

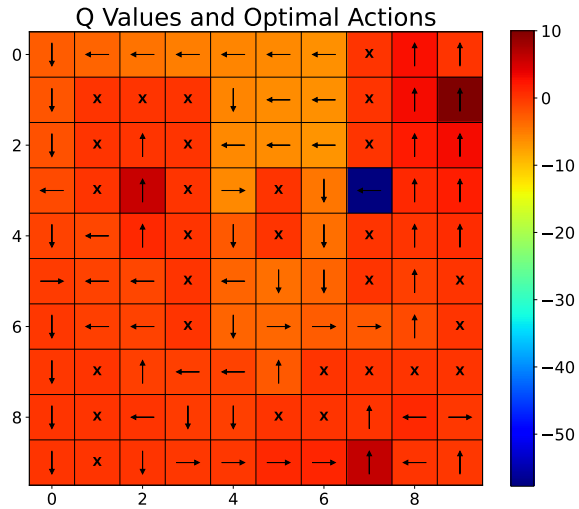
| | |
|---|---|
| Reward | -4 |
| Path taken | [36, 46, 56, 66, 68, 58, 48, 39, 29, 19, 9] |



# 6 Problem 4

For each of the experiments, provide a written description of the similarities and differences between the best policy learnt by SARSA and $Q-$learning along with the hyperparameters, justifications and plots requested above. Your description should include an explanation for the behaviour of the policy learnt by each algorithm.

---

We infer the following deductions, that has been backed up with some reasonable explanations:

1. **SARSA vs $Q-$Learning:** Note that each of SARSA and $Q-$Learning generates the optimal reward with the best possible path in the Stochasticity variant 1, when the situation is non-windy and deterministic for each of the policies, that is Softmax and $\epsilon-$greedy. In a deterministic environment with no stochasticity, both SARSA (State-Action-Reward-State-Action) and Q-learning algorithms are expected to converge to the same optimal policy and generate the same result. This is because the core update rules for both algorithms are based on the same underlying principle of estimating the optimal action-value function.

   On the other hand, for a stochastic environment, $Q-$Learning outperforms SARSA in most cases of the Variant 2 and Variant 3. In stochastic environments, where the outcomes of actions are uncertain and subject to randomness, Q-learning often performs better than SARSA. The reason lies in the fundamental difference in the update mechanisms of these two algorithms. Q-learning is an off-policy algorithm, meaning it updates its Q-values based on the maximum expected future reward, regardless of the actual action taken. The use of $\max_{a'} Q(s', a')$ makes Q-learning less sensitive to the exploratory actions taken during learning. SARSA, on the other hand, is an on-policy algorithm. It updates its Q-values based on the actual action taken and the subsequent action chosen according to its policy. Since SARSA updates its Q-values based on the actions it selects during exploration, it can be more influenced by the stochasticity in the environment. In a stochastic environment, where the best action may not always result in the maximum observed reward due to randomness, Q-learning tends to perform better because it effectively learns to generalize and select actions that lead to higher expected future rewards.

2. **Softmax vs $\epsilon-$greedy:** Note that each of softmax and $\epsilon-$greedy generates the same output for the variant 1. But for a windy environment or a stochastic agent, softmax performs quite better as compared to $\epsilon-$greedy. The reasons can be described as follows:

   (a) **Adaptive Exploration:** Softmax allows for adaptive exploration by adjusting the exploration probability based on the uncertainty in Q-values. This adaptability can be beneficial in handling stochastic environments.

   (b) **Smooth Exploration:** Softmax provides a smooth, probabilistic exploration strategy. This allows the agent to explore multiple actions with varying probabilities, which can be advantageous in environments where the reward distribution is uncertain. It helps the agent to consider a range of actions rather than being stuck with a deterministic exploration policy. This smoothness of softmax over $\epsilon-$greedy can be observed from the two plots, that are – Reward $\times$ Episodes curve and Path-length $\times$ Episodes curve for any given experiment of those 12 cases.

   (c) **Avoiding Greedy Traps:** In stochastic environments, blindly selecting the action with the highest Q-value (as in Epsilon-Greedy) may lead to suboptimal behaviour. Softmax

exploration, by considering the entire distribution of Q-values, is less prone to getting stuck in greedy traps and is more likely to explore diverse actions.

In other words, Softmax exploration tends to outperform Epsilon-Greedy in stochastic environments because it offers a more adaptive and nuanced approach to exploration, allowing the agent to better navigate uncertainty and discover the true characteristics of the environment.

3. **The nature of the reward curve and the path length curve in each variant of stochasticity:** For the non-windy deterministic variant, both of these plots enjoy less standard deviation and stochasticity, on the other hand in the windy or the stochastic variant, they experience in between spikes and sometimes more standard deviation due to the quite a few reasons such as –

   (a) **Inherent Variability:** Randomness can lead to fluctuations in rewards even when the agent follows the same policy. This variability can result in occasional spikes and higher standard deviation in the reward curve.

   (b) **Randomness in State Transitions:** For Variant 2 (aka non-windy stochastic), we observe relatively larger spikes (and standard deviation) in the plot (then Variant 3 – windy deterministic and finally the Variant 1 – non-windy deterministic). Here, the stochasticity in state transitions can contribute to variability.

4. **The behaviour of the agent post-training:** Please refer to Inference on the post-training path under Problem 2.

# 7 REFERENCES

[1] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 2nd edition, 2018.