CSE574-D: Introduction to Machine Learning
Assignment 3: Defining and Solving Reinforcement Learning Task

Ashmita Pandey
University at Buffalo
pandey7(50485164)

December 8, 2023

Contents

1 Part I: Define an RL Environment **2**
1.1 Environment Scenario . . . 2
1.2 RL Environment Definition . . . 2
1.3 Random Agent Simulation . . . 2
1.4 Visualization of the Environment . . . 2
1.5 Safety in AI . . . 3

2 Part II: Implement SARSA **4**
2.1 SARSA Implementation . . . 4
2.1.1 Initial Q-Table . . . 4
2.1.2 Trained Q-Table . . . 4
2.1.3 Evaluation Results . . . 4
2.1.4 Total Reward Per Episode . . . 4
2.1.5 Epsilon Decay . . . 5
2.1.6 Visualization of the Environment . . . 5
2.2 Hyperparameter Tuning . . . 6
2.2.1 Selected Hyperparameters and Rationale . . . 7
2.2.2 Evaluation of Hyperparameters . . . 7
2.2.3 Analysis of Results . . . 7
2.2.4 Plots of Tuned Hyperparameters . . . 7

3 Part III: Implement Double Q-learning **9**
3.1 Double Q-learning Implementation . . . 9
3.1.1 Initial Q-Tables . . . 9
3.1.2 Trained Q-Tables . . . 9
3.1.3 Evaluation Results . . . 9
3.1.4 Total Reward Per Episode . . . 9
3.1.5 Epsilon Decay . . . 10
3.2 Hyperparameter Tuning for Double Q-learning . . . 10
3.2.1 Selected Hyperparameters and Rationale . . . 10
3.2.2 Evaluation of Hyperparameters . . . 11
3.2.3 Analysis of Results . . . 11
3.2.4 Plots of Tuned Hyperparameters . . . 11
3.3 Comparison of SARSA and Double Q-learning . . . 12

4 Bonus Task: Implementing n-step Bootstrapping **12**
4.1 Implementation of n-step SARSA . . . 13
4.1.1 Results and Analysis . . . 13
4.1.2 Comparison with Base SARSA . . . 13

Introduction

Reinforcement Learning (RL) is a crucial area of machine learning where an agent learns to make decisions by interacting with an environment. Through this interaction, the agent learns to perform tasks by trial and error, guided by rewards or penalties. The core idea is to find a strategy, known as a policy, that maximizes the cumulative reward for the agent over time.

This assignment, focuses on the practical application of RL principles. It requires the definition and resolution of a reinforcement learning environment following Gym standards, providing hands-on experience with the dynamics of RL systems.

1 Part I: Define an RL Environment

1.1 Environment Scenario

Our chosen scenario for the grid world is a Lawnmower Grid World, where the lawnmower aims to collect batteries for positive rewards and avoid rocks for negative rewards. The grid world is a 4×4 environment with the following specifications:

- States: The environment includes 16 discrete states, $S1 = (0, 0)$ to $S16 = (3, 3)$.
- Actions: The lawnmower has a set of 4 possible actions: {Up, Down, Right, Left}.
- Rewards: The lawnmower receives rewards of {-5, -6, +5, +6} for encountering rocks or batteries, respectively.
- Objective: The main objective is to navigate through the grid to collect as many batteries as possible while avoiding rocks, within a set number of timesteps.

1.2 RL Environment Definition

The environment is defined in Python using the Gym library, with the following methods implemented to simulate the reinforcement learning setup:

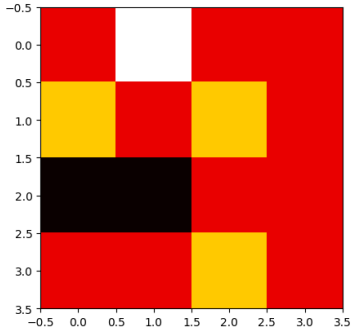
```
def __init__(self):
    # Initializes the environment with action and observation space.

def step(self, action):
    # Executes one timestep within the environment with the given action.

def reset(self):
    # Resets the environment to an initial state.

def render(self):
    # Visualizes the current state of the environment.
```

1.3 Random Agent Simulation

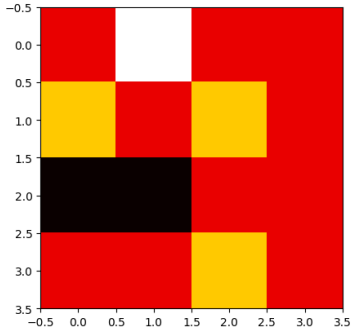
A random agent was run for at least 10 timesteps to validate the correct definition of the environment logic. The agent's state, chosen action, reward, and a visualization of the grid world were provided for each timestep.

1.4 Visualization of the Environment

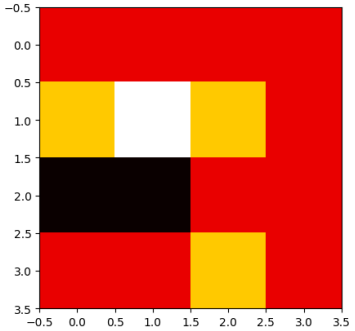
The visualization of the environment after each action taken by the random agent is shown below:



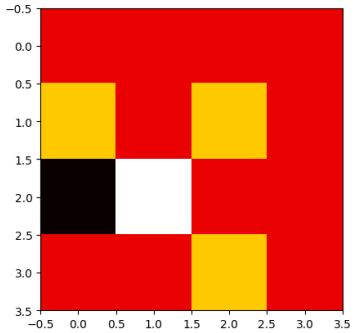
(a) State: [0, 1], Action: 2, Reward: 0.0



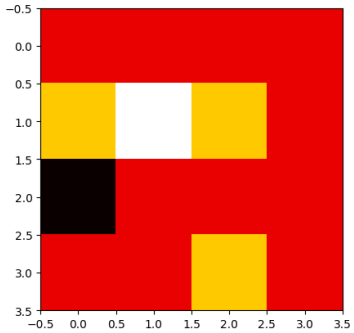
(b) State: [0, 1], Action: 0, Reward: 0.0



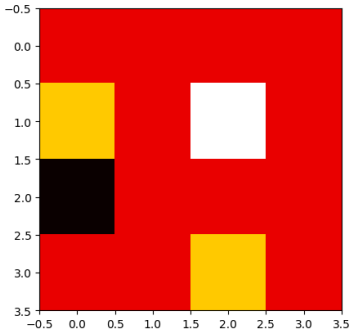
(c) State: [1, 1], Action: 1, Reward: 0.0



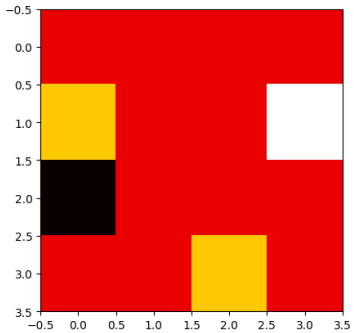
(d) State: [2, 1], Action: 1, Reward: -5.0



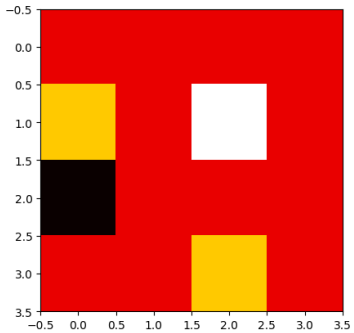
(e) State: [1, 1], Action: 0, Reward: 0.0



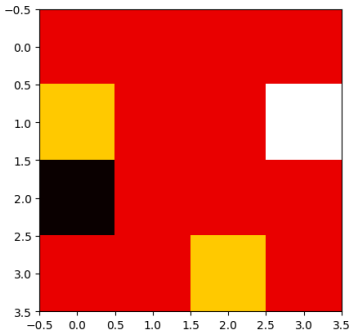
(f) State: [1, 2], Action: 2, Reward: 5.0



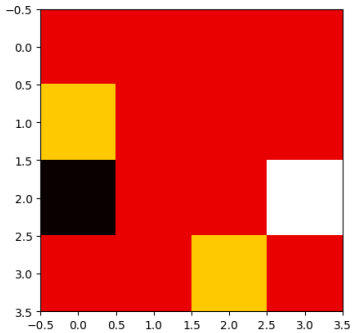
(g) State: [1, 3], Action: 2, Reward: 0.0



(h) State: [1, 2], Action: 3, Reward: 0.0



(i) State: [1, 3], Action: 2, Reward: 0.0



(j) State: [2, 3], Action: 1, Reward: 0.0

1.5 Safety in AI

Ensuring safety within the RL environment involves guaranteeing that the agent operates within the defined state space and only selects valid actions. This is achieved through the following measures:

- The action space is limited to predefined actions, and any action taken by the agent is checked against this set.
- The state transitions are bounded within the grid limits, preventing the agent from moving to an undefined state.
- Rewards and penalties are structured to guide the agent towards safe and goal-oriented behavior.
- The environment's 'step' function includes logic to handle the termination of an episode, ensuring the agent does not continue beyond the maximum number of steps.

2 Part II: Implement SARSA

In Part II of our project, we implemented the SARSA (State-Action-Reward-State-Action) algorithm, an on-policy reinforcement learning method, to solve the Lawnmower Grid World environment established in Part I. SARSA algorithm uses the Q-learning update rule and an exploration-exploitation strategy to balance between learning new actions and exploiting known rewards.

2.1 SARSA Implementation

The SARSA algorithm was applied to the Lawnmower Grid World environment. The initial and trained Q-tables were generated, and the agent's performance was evaluated over 1000 episodes.

2.1.1 Initial Q-Table

The Q-table was initialized with zeros, representing the lack of prior knowledge before the agent begins interacting with the environment.

```
Initial Q-table:
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 ...
 [0. 0. 0. 0.]]
```

2.1.2 Trained Q-Table

After training, the Q-table showed the learned values for each state-action pair, guiding the agent to take actions that would maximize future rewards.

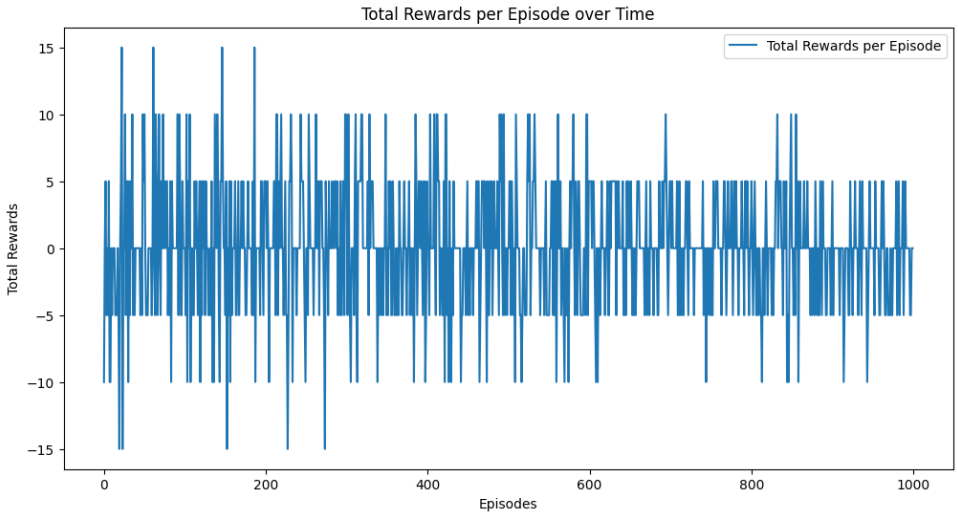
```
Trained Q-table:
[[-2.34535097e-01 -2.20437750e+00 -1.33671644e+00 -1.22808138e+00]
 ...
 [ 0.00000000e+00  0.00000000e+00  9.38369704e-05 -2.34839554e-03]]
```

2.1.3 Evaluation Results

The agent was evaluated over 10 episodes, choosing only greedy actions from the learned policy. The rewards per episode were plotted to assess the performance.

2.1.4 Total Reward Per Episode

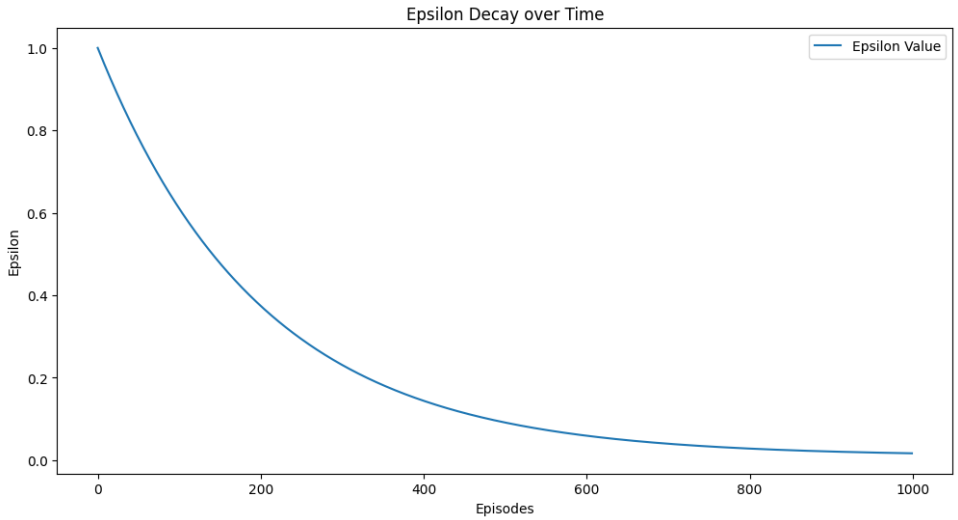
The following graph shows the total rewards obtained by the agent in each episode throughout the training process.



(a) Total Reward Per Episode Over Time

2.1.5 Epsilon Decay

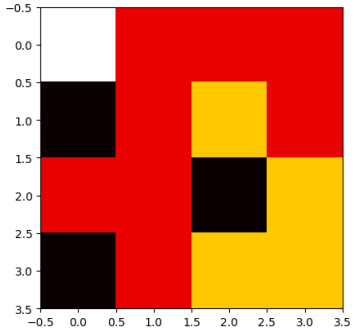
The epsilon decay plot demonstrates the agent's transition from exploration to exploitation over time.



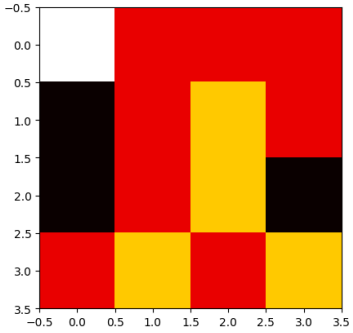
(a) Epsilon Decay Over Time

2.1.6 Visualization of the Environment

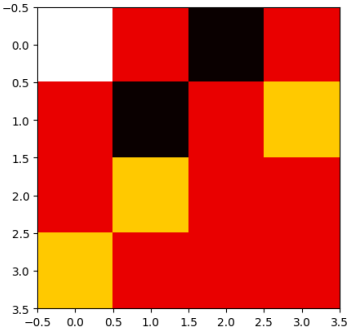
The visualization of the environment after each action taken by the random agent is shown below:



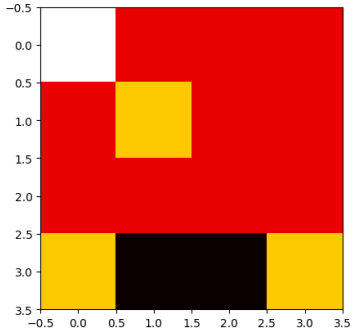
(a) EPISODE = 0 Score = -5.0



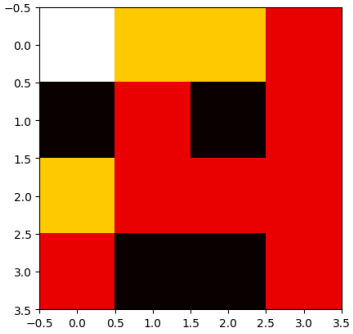
(b) EPISODE = 1 Score = 0.0



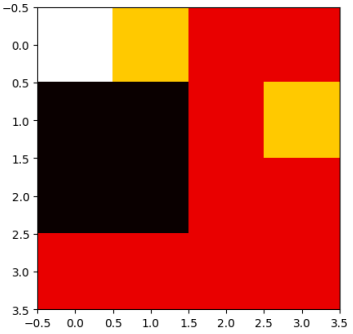
(c) EPISODE = 2 Score = 0.0



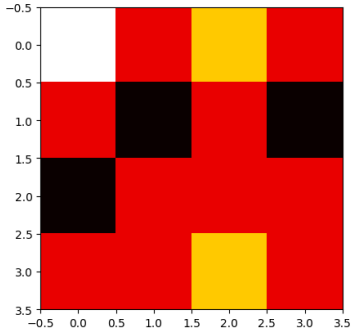
(d) EPISODE = 3 Score = 0.0



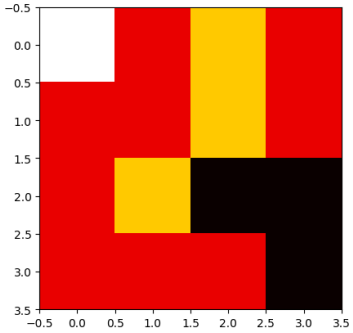
(e) EPISODE = 4 Score = 0.0



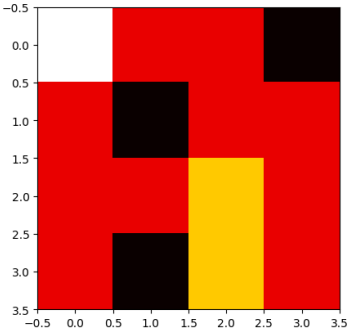
(f) EPISODE = 5 Score = 0.0



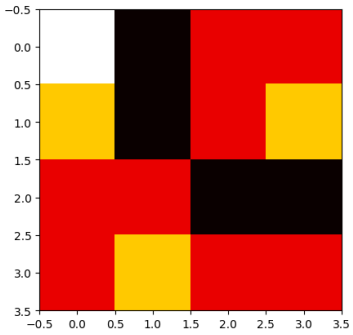
(g) EPISODE = 6 Score = 5.0



(h) EPISODE = 7 Score = 5.0



(i) EPISODE = 8 Score = 0.0



(j) EPISODE = 9 Score = 0.0

2.2 Hyperparameter Tuning

To optimize the SARSA agent's learning performance in the Lawnmower Grid World environment, we focused on tuning two critical hyperparameters: the discount factor γ and the epsilon decay rate. The discount factor influences how much the agent prioritizes future rewards over immediate ones, while the epsilon decay rate controls the rate at which the agent shifts from exploring the environment to exploiting its learned policy.

2.2.1 Selected Hyperparameters and Rationale

We chose to explore the following values for each hyperparameter:

- Discount factor γ: [0.8, 0.9, 0.99] — to assess the agent's sensitivity to immediate versus future rewards.
- Epsilon decay rate: [0.005, 0.01, 0.02] — to find a balance between exploration and exploitation over time.

These values were selected to represent a range from conservative to aggressive in terms of how quickly the agent would start relying on its learned policy versus continuing to explore the state space.

2.2.2 Evaluation of Hyperparameters

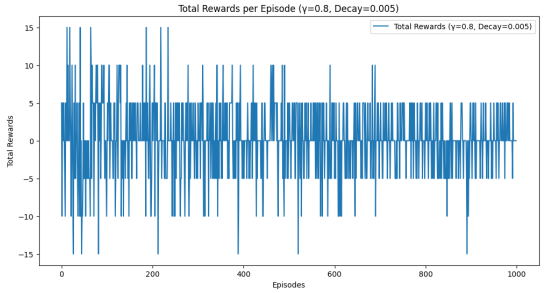
The SARSA algorithm was executed with each combination of γ and decay rates, resulting in a total of nine distinct configurations. Each configuration was run for 1000 episodes to ensure adequate learning time. The evaluation focused on the consistency and magnitude of the rewards obtained by the agent as it interacted with the environment.

2.2.3 Analysis of Results

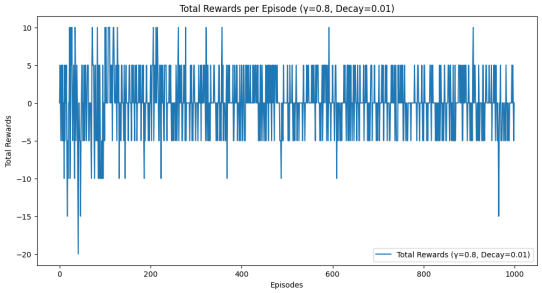
The results indicated varied performance across different settings. Lower discount factors tended to make the agent myopic, focusing more on immediate rewards, whereas higher discount factors encouraged the agent to consider future rewards more strongly. As for the epsilon decay rate, faster decay rates led to quicker convergence in policy but at the potential cost of suboptimal exploration.

2.2.4 Plots of Tuned Hyperparameters

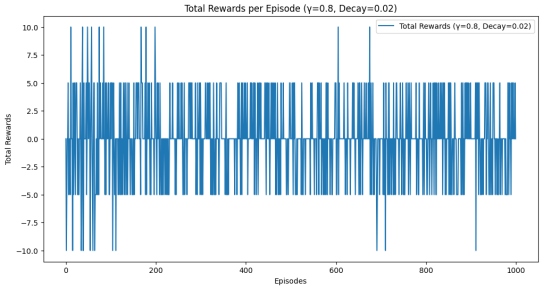
The total reward per episode was plotted for each hyperparameter combination to visualize the learning progression and the agent's performance stability.



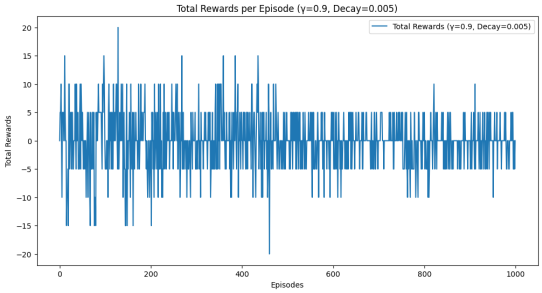
(a) $\gamma = 0.8$ and Decay Rate $= 0.005$



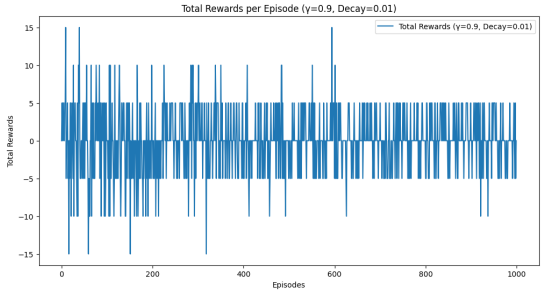
(b) $\gamma = 0.8$ and Decay Rate $= 0.01$



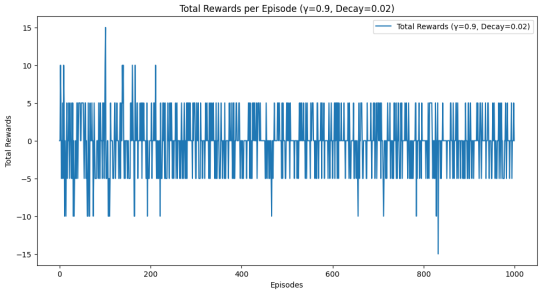
(c) $\gamma = 0.8$ and Decay Rate $= 0.02$



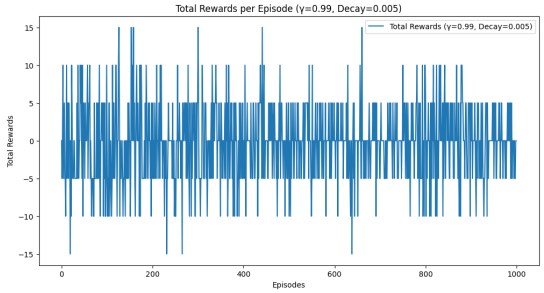
(d) $\gamma = 0.9$ and Decay Rate $= 0.005$



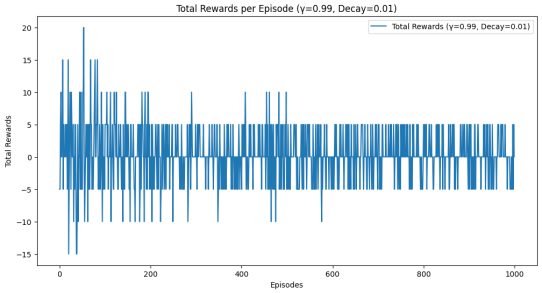
(e) $\gamma = 0.9$ and Decay Rate $= 0.01$



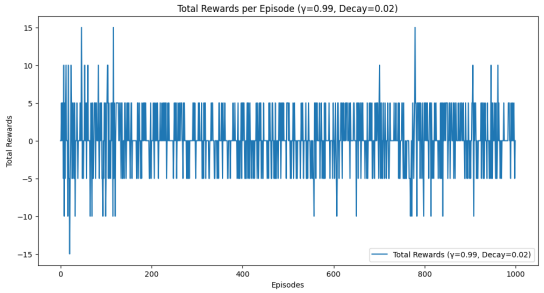
(f) $\gamma = 0.9$ and Decay Rate $= 0.02$



(g) $\gamma = 0.99$ and Decay Rate $= 0.005$



(h) $\gamma = 0.99$ and Decay Rate $= 0.01$



(i) $\gamma = 0.99$ and Decay Rate $= 0.02$

The plots show the agent's adaptation to the environment under different learning dynamics. A higher γ and lower decay rate often led to a smoother learning curve, whereas lower γ and higher decay rate resulted in more erratic performance.

Based on the conducted experiments, a discount factor of $\gamma = 0.9$ and an epsilon decay rate of 0.005

provided the most consistent and high rewards, indicating an effective balance between short-term and long-term reward maximization and between exploration and exploitation.

The implementation of SARSA demonstrated the significance of tuning hyperparameters in reinforcement learning. While some hyperparameter settings led to improved learning and stability, others did not show a significant impact on the agent's performance.

3 Part III: Implement Double Q-learning

Building on Part II, we implemented the Double Q-learning algorithm, an off-policy reinforcement learning method, for the Lawnmower Grid World environment. Double Q-learning uses two Q-tables to reduce the overestimation bias of traditional Q-learning and an epsilon-greedy strategy for action selection.

3.1 Double Q-learning Implementation

The Double Q-learning algorithm was applied to the Lawnmower Grid World environment. Initial and trained Q-tables were generated, and the agent's performance was evaluated over 1000 episodes with a focus on the setup that returned the best results.

3.1.1 Initial Q-Tables

The Q-tables were initialized with zeros, representing the agent's initial lack of knowledge.

3.1.2 Trained Q-Tables

After training, the Q-tables showed the learned values for each state-action pair. These tables guide the agent to take actions that maximize future rewards while minimizing the overestimation bias.

3.1.3 Evaluation Results

The agent was evaluated over 10 episodes using only greedy actions from the learned policy, and the rewards per episode were plotted.

3.1.4 Total Reward Per Episode

The following graph shows the total rewards obtained by the agent in each episode throughout the training process.

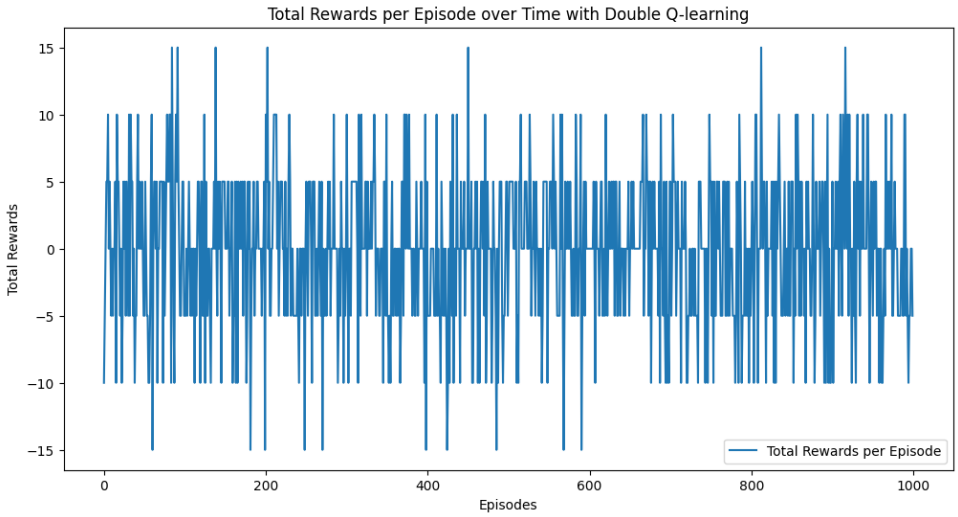


Figure 6: Total Reward Per Episode Over Time with Double Q-learning

3.1.5 Epsilon Decay

The epsilon decay plot demonstrates the agent's transition from exploration to exploitation over time.

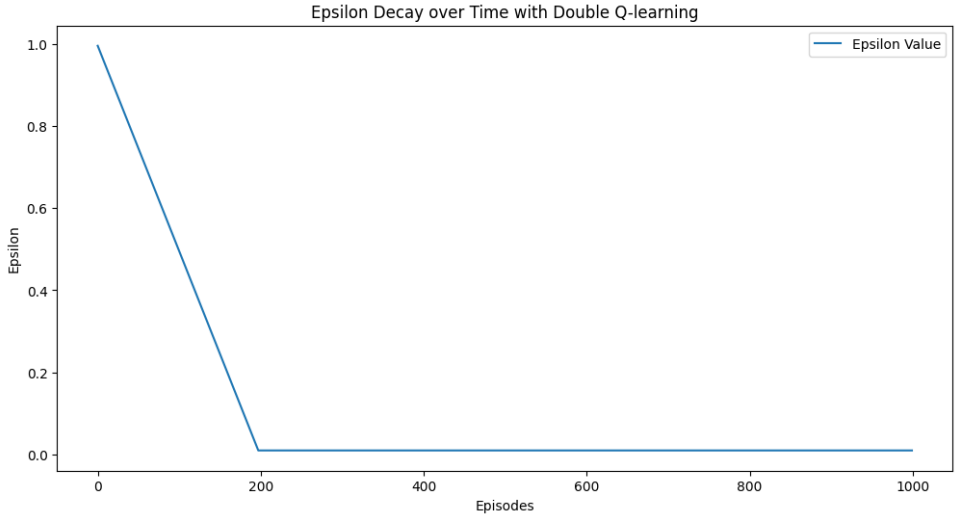


Figure 7: Epsilon Decay Over Time with Double Q-learning

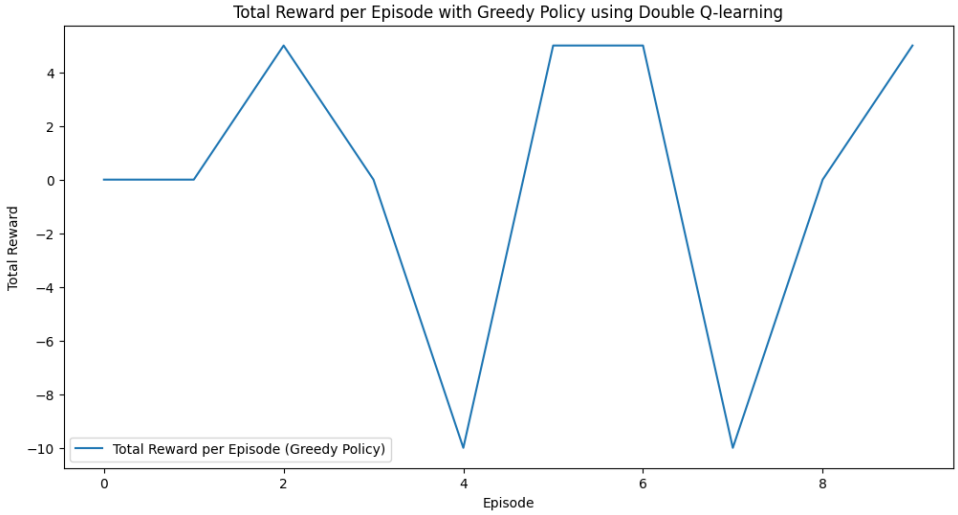


Figure 8: Total Reward Per Episode with Greedy Policy using Double Q-learning

3.2 Hyperparameter Tuning for Double Q-learning

For Double Q-learning, we focused on tuning two hyperparameters: the discount factor γ and the epsilon decay rate. These parameters are crucial as they determine the balance between short-term and long-term rewards and the shift from exploration to exploitation, respectively.

3.2.1 Selected Hyperparameters and Rationale

The discount factor γ was selected to be tested at values of 0.8, 0.9, and 0.99 to understand its effect on the agent's foresight in valuing future rewards. The epsilon decay rate was chosen to be examined at 0.005, 0.01, and 0.02 to find the optimal rate at which the agent transitions from exploring the environment to exploiting its learned strategy.

3.2.2 Evaluation of Hyperparameters

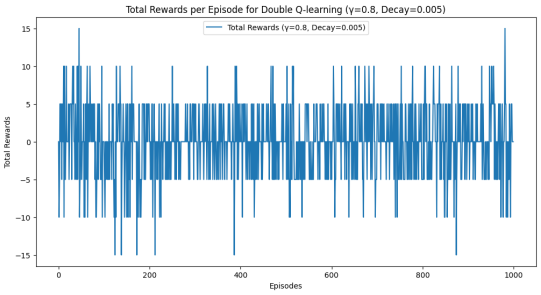
We evaluated each set of hyperparameters over 1000 episodes, measuring the total reward per episode to gauge the agent's performance under different configurations.

3.2.3 Analysis of Results

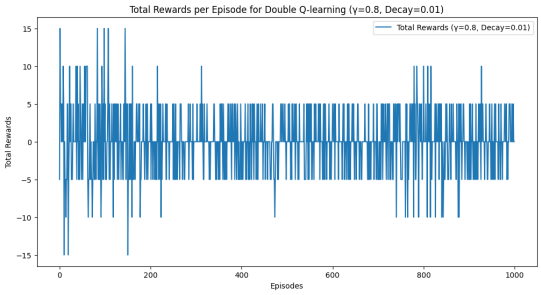
The analysis of the results aimed to determine the optimal balance between exploration and exploitation that yields the highest rewards. We looked for trends in the rewards history, such as increasing average rewards or decreased variability in rewards, to identify the most effective hyperparameter values.

3.2.4 Plots of Tuned Hyperparameters

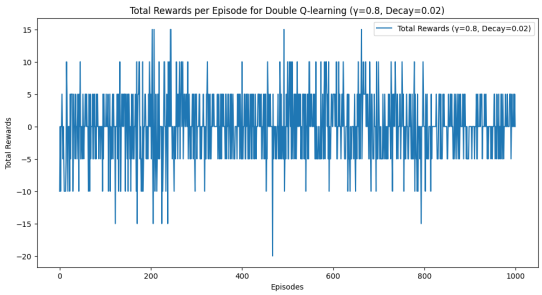
The following plots illustrate the agent's performance across the different sets of hyperparameters, highlighting the impact of each configuration on the learning process.



(a) $\gamma = 0.8$ and Decay Rate $= 0.005$



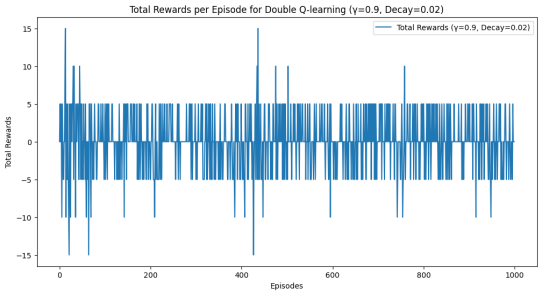
(b) $\gamma = 0.8$ and Decay Rate $= 0.01$



(c) $\gamma = 0.8$ and Decay Rate $= 0.02$

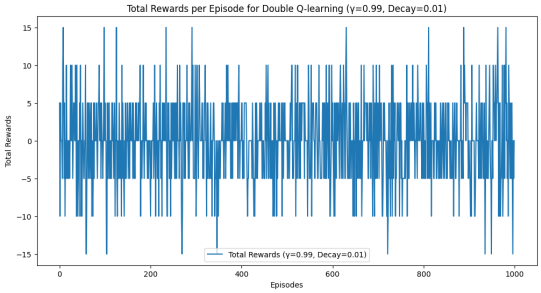
(d) $\gamma = 0.9$ and Decay Rate $= 0.005$

(e) $\gamma = 0.9$ and Decay Rate $= 0.01$

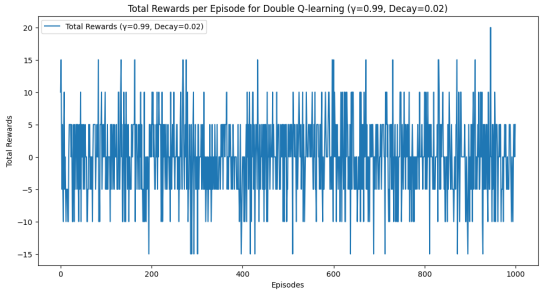


(f) $\gamma = 0.9$ and Decay Rate $= 0.02$

(a) $\gamma = 0.99$ and Decay Rate $= 0.005$



(b) $\gamma = 0.99$ and Decay Rate $= 0.01$



(c) $\gamma = 0.99$ and Decay Rate $= 0.02$

3.3 Comparison of SARSA and Double Q-learning

The comparison between SARSA and Double Q-learning revealed distinct learning behaviors. While SARSA, being an on-policy method, showed a more conservative learning approach, Double Q-learning exhibited a different pattern due to its off-policy nature and dual Q-tables, which are designed to mitigate the overestimation of Q-values.

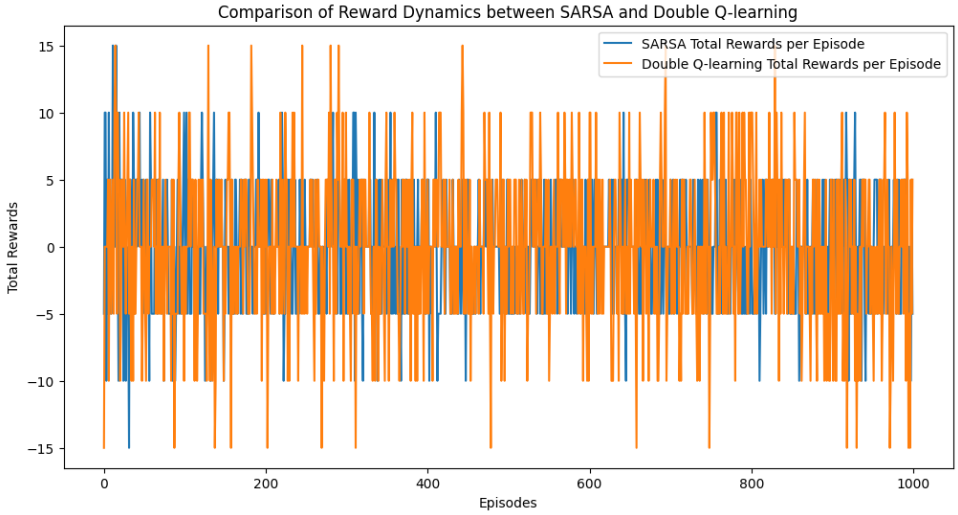


Figure 11: Comparison of Reward Dynamics between SARSA and Double Q-learning

4 Bonus Task: Implementing n-step Bootstrapping

In this section, we extended the SARSA algorithm to incorporate n-step Bootstrapping, specifically implementing a 2-step SARSA. This method aims to strike a balance between the immediate and delayed rewards by considering multiple future steps when updating Q-values.

4.1 Implementation of n-step SARSA

The n-step SARSA algorithm was implemented by modifying the base SARSA algorithm to consider the rewards and actions of the next ‘n‘ steps. We chose ‘n = 2‘ for this implementation. The Q-table was updated considering the sum of the rewards for the next 2 steps, weighted by the discount factor γ.

4.1.1 Results and Analysis

The agent’s performance with 2-step SARSA was evaluated over 1000 episodes. The total rewards per episode were recorded and compared to the base SARSA algorithm to assess the effectiveness of n-step Bootstrapping.

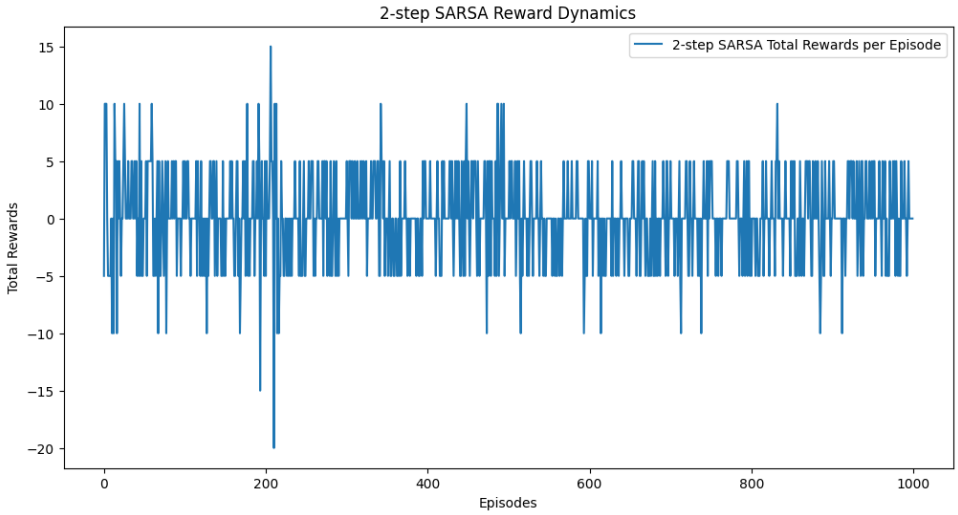


Figure 12: Total Rewards per Episode with 2-step SARSA

4.1.2 Comparison with Base SARSA

The results of 2-step SARSA were compared with the base SARSA algorithm. This comparison aimed to understand how considering multiple future steps affects the learning and decision-making process of the agent.

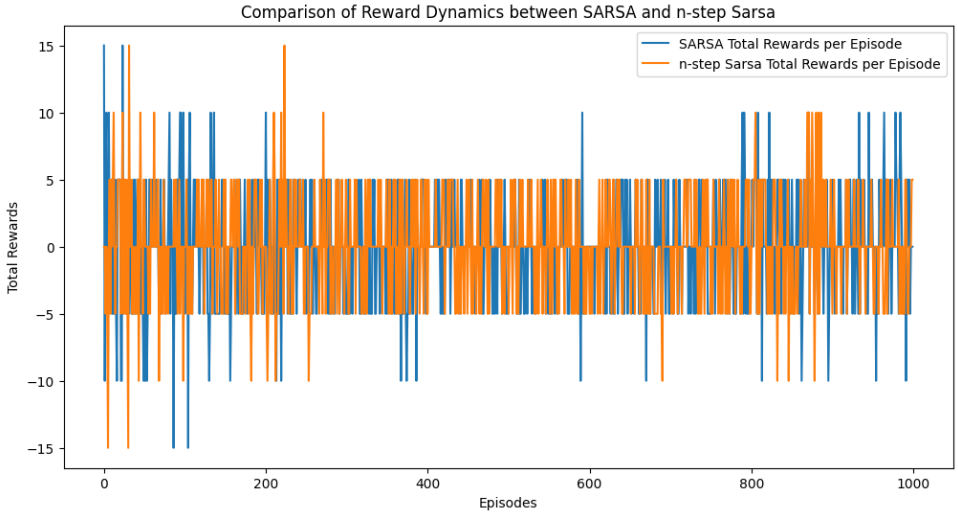


Figure 13: Comparison of Reward Dynamics between 2-step SARSA and Base SARSA

References

- **Gymnasium**: A toolkit for developing and comparing reinforcement learning algorithms. *Gymnasium*. Available at: `https://gymnasium.farama.org/`
- **Numpy**: Scientific computing with Python. *NumPy Documentation*. Available at: `https://numpy.org/doc/1.26/`
- **Matplotlib**: Visualization with Python. *Matplotlib Documentation*. Available at: `https://matplotlib.org/stable/index.html`
- **Sarsa**: A detailed explanation of the SARSA algorithm. *GeeksforGeeks*. Available at: `https://www.geeksforgeeks.org/sarsa-reinforcement-learning/`
- **Double Q-learning**: Hado van Hasselt. Double Q-learning. *Advances in Neural Information Processing Systems 23*, 2010. Available at: `https://proceedings.neurips.cc/paperfiles/paper/2010/file/091d584fced301b442654dd8c23b3fc9-Paper.pdf`
- **n-step Bootstrapping**: An introduction to n-step bootstrapping in reinforcement learning. *Towards Data Science*. Available at: `https://towardsdatascience.com/introduction-to-reinforcement-learni`