**ASSIGNMENT 1**

**Defining & Solving RL Environments**

**1)Describe the deterministic and stochastic environments, which were defined (set of actions/states/rewards, main objective, etc).**

Deterministic Environment:

The deterministic environment models a warehouse robot that follows a grid-world structure with fixed movement rules and obstacles. The environment operates with the following characteristics:

States: The state space is defined by the robot's position (x, y) and whether it is carrying an item (binary variable: 0 or 1).

Actions: The agent has 6 discrete actions:

* 1. Move Up
  2. Move Down
  3. Move Left
  4. Move Right
  5. Pick-up the package (if at the item's position)
  6. Drop-off the package (if at the drop-off location)

Obstacles: The grid contains static obstacles that block movement.

Rewards:

-1 per step (encouraging efficiency)

-20 penalty for hitting an obstacle

+25 reward for picking up the item

+100 reward for successfully dropping off the item

Main Objective: The agent must navigate from the start position, pick up an item at a fixed location, and drop it off at another fixed location while avoiding obstacles. The optimal policy is to minimize steps and avoid obstacles.

Stochastic Environment (Stochastic\_warehouse Class)

The stochastic version introduces randomness in the agent’s movement and dynamic obstacles, making navigation more unpredictable.

States: Similar to the deterministic environment: robot position (x, y) and whether it's carrying an item.

Actions: The same 6 discrete actions as before.

Stochastic Behavior:

With 10% probability, the robot's movement fails, meaning the intended action may not execute.

This introduces uncertainty, making planning more challenging.

Rewards:

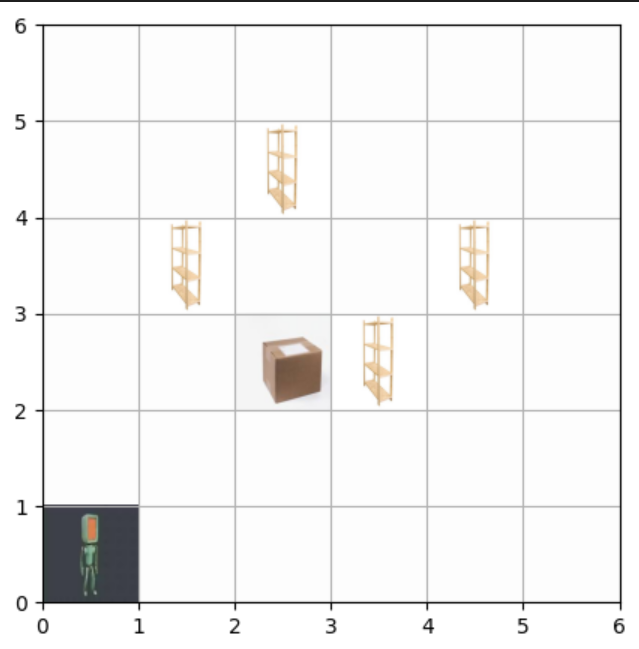
Similar reward structure as in the deterministic environment.

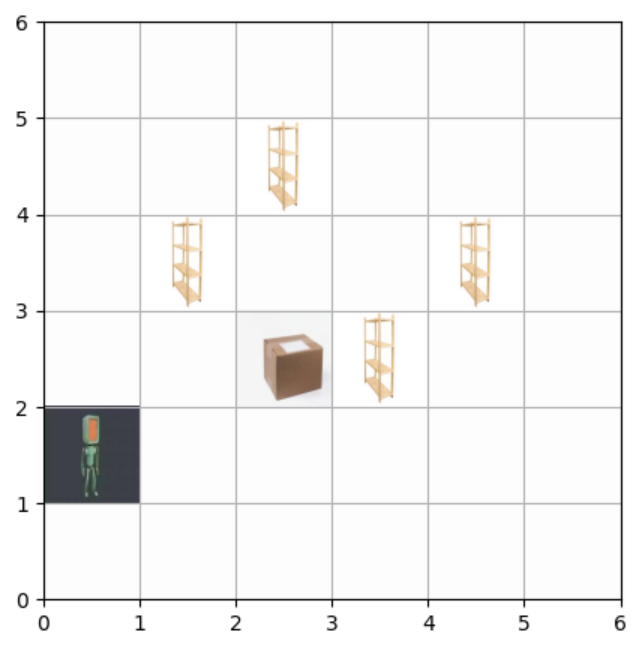
However, because of stochastic movement, the agent may receive penalties unintentionally.

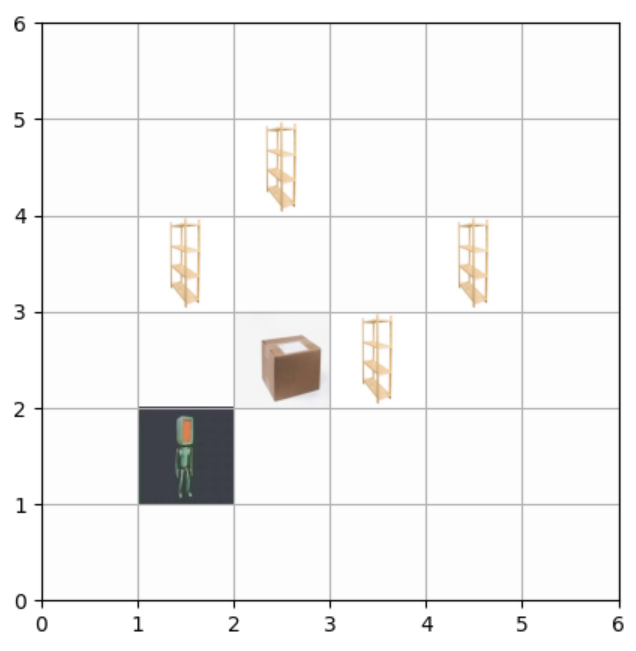
Main Objective: The goal remains the same (pick up and drop off an item), but due to random failures and moving obstacles, the agent must adapt its policy to account for unexpected setbacks.

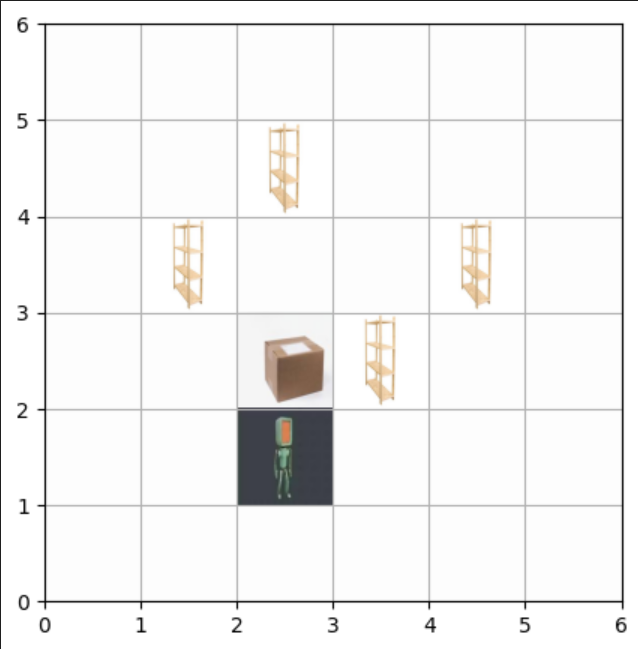
**2) Provide visualizations of your environments.**

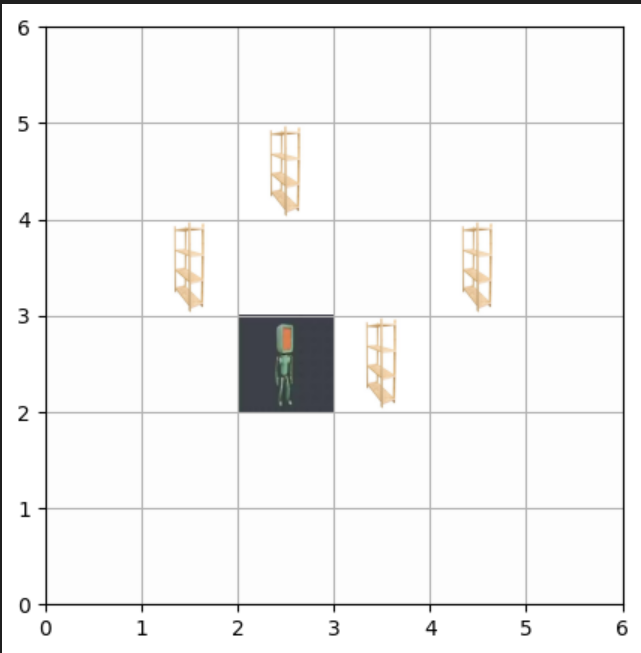
Deterministic environment:

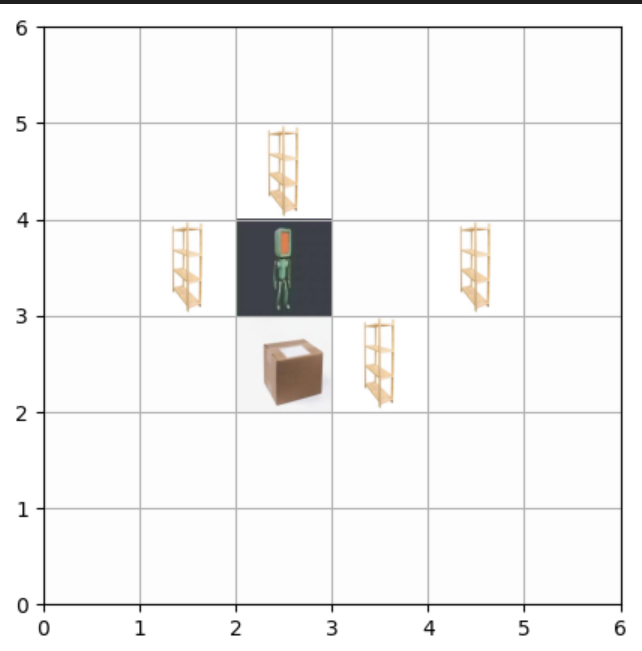
****

****

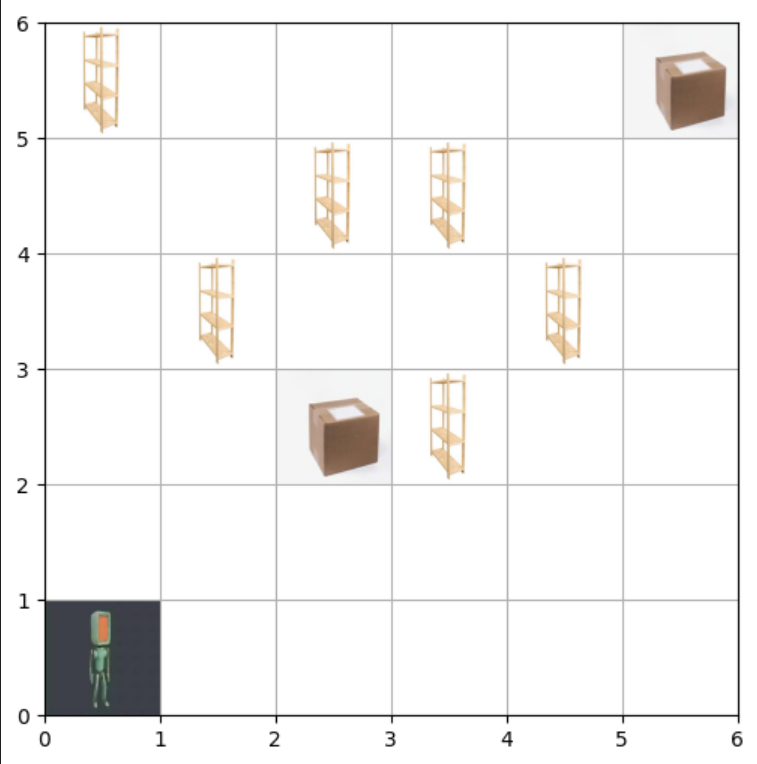
****

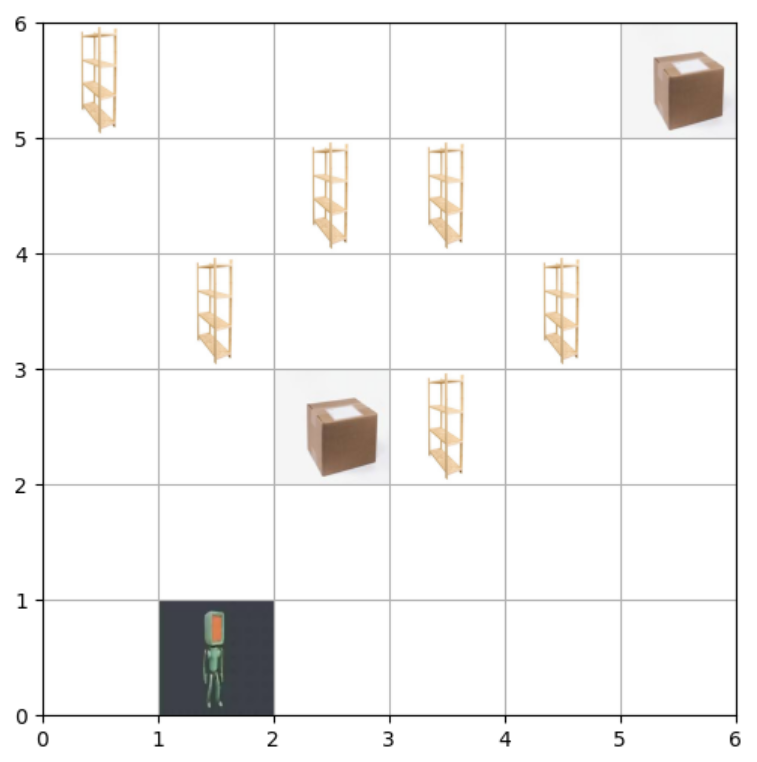
****

****

****

**STOCHASTIC ENVIRONMENT:**

****

****

I have provided different pictures of my environment above.

**3) How did you define the stochastic environment?**

The stochastic environment was defined by introducing randomness in movement and dynamic obstacles to make navigation less predictable. The key modifications from the deterministic environment include:

Stochastic Movement:

With a 10% probability, the agent's movement fails, meaning the intended action does not execute.

This simulates real-world uncertainty, such as slipping, sensor errors, or mechanical failures.

State Representation:

The state still consists of the robot’s position (x, y) and whether it is carrying an item (0 or 1).

1. Action Set (Same as Deterministic Environment):

6 discrete actions: Up, Down, Left, Right, Pick-up, Drop-off.

1. Reward Structure (Same as Deterministic Environment):

-1 per step (to encourage efficiency).

-20 penalty for colliding with obstacles.

+25 reward for successfully picking up the item.

+100 reward for successfully dropping off the item.

**4) What is the difference between the deterministic and stochastic environments**

Differences:

**Deterministic Environment:**

1) The agent's actions always lead to the expected outcome.  
2) Fixed obstacles remain in the same position throughout.  
3) No randomness in movement or state transitions.  
4) The agent can follow a fixed optimal path.  
5) Predictable environment, making it easier to solve.

**Stochastic Environment:**

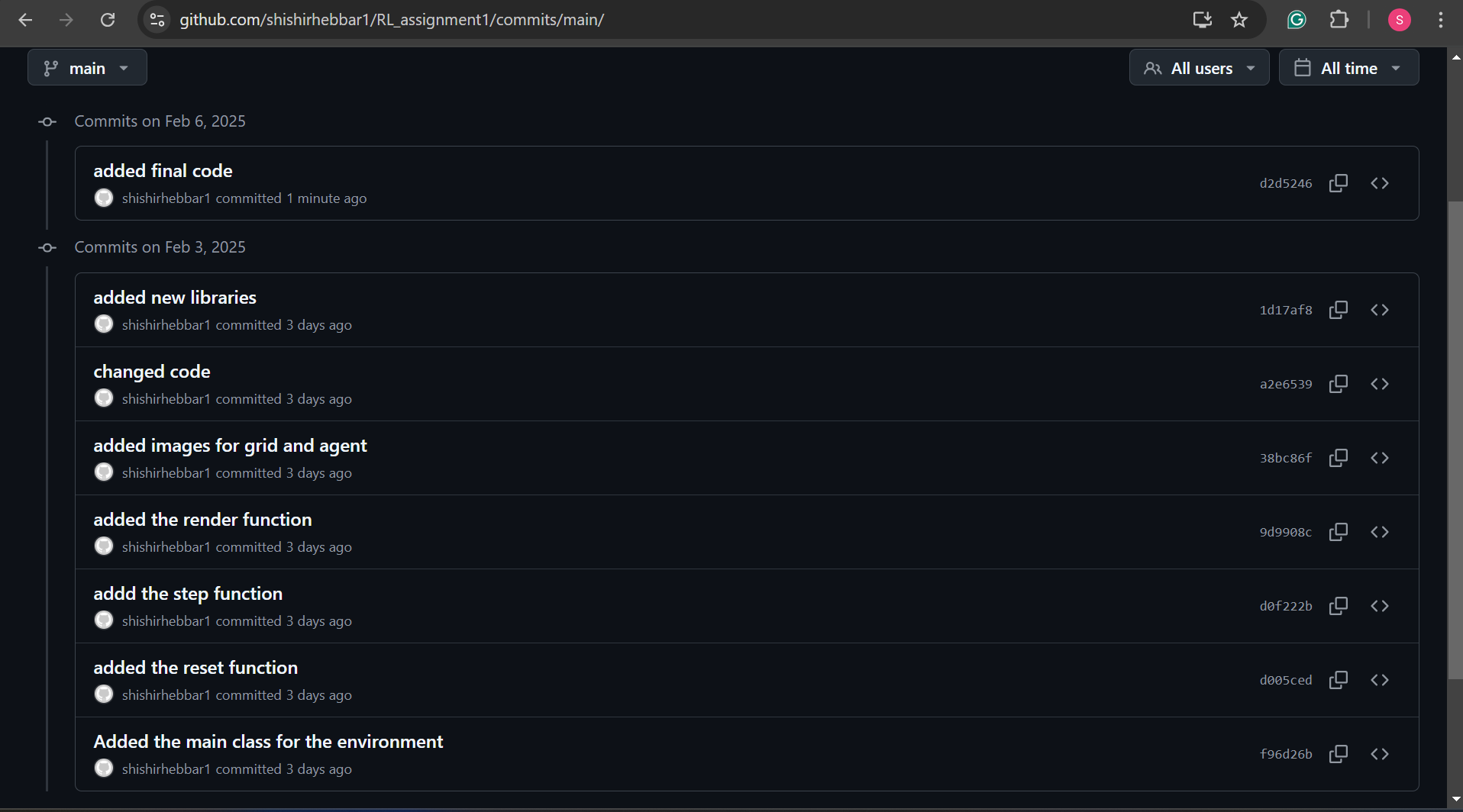
1) 10% chance of movement failure, adding randomness.  
2) Includes dynamic obstacles that change positions over time.  
3) The agent may not always move as expected, requiring adaptability.  
4) The same action may lead to different outcomes in different runs.  
5) More realistic and challenging, requiring flexible strategies.

**5) Safety in AI: Write a brief review (∼ 5 sentences) explaining how you ensure the safety of your environments. E.g. how do you ensure that agent choose only actions that are allowed, that agent is navigating within defined state-space, etc**

Ensuring safety in AI environments involves setting strict constraints on the agent’s actions and state transitions. The environment defines a finite grid with clear boundaries, preventing the agent from moving outside the allowed space. Actions that would lead to invalid states (e.g., moving into obstacles) are handled by reverting the agent's position and applying a negative reward to discourage such behavior. Additionally, in the stochastic environment, randomness is controlled to ensure that transitions remain within safe limits while still introducing uncertainty. By implementing these safeguards, the agent learns to navigate efficiently while avoiding unsafe or undefined behaviors.

Github link: <https://github.com/shishirhebbar1/RL_assignment1>

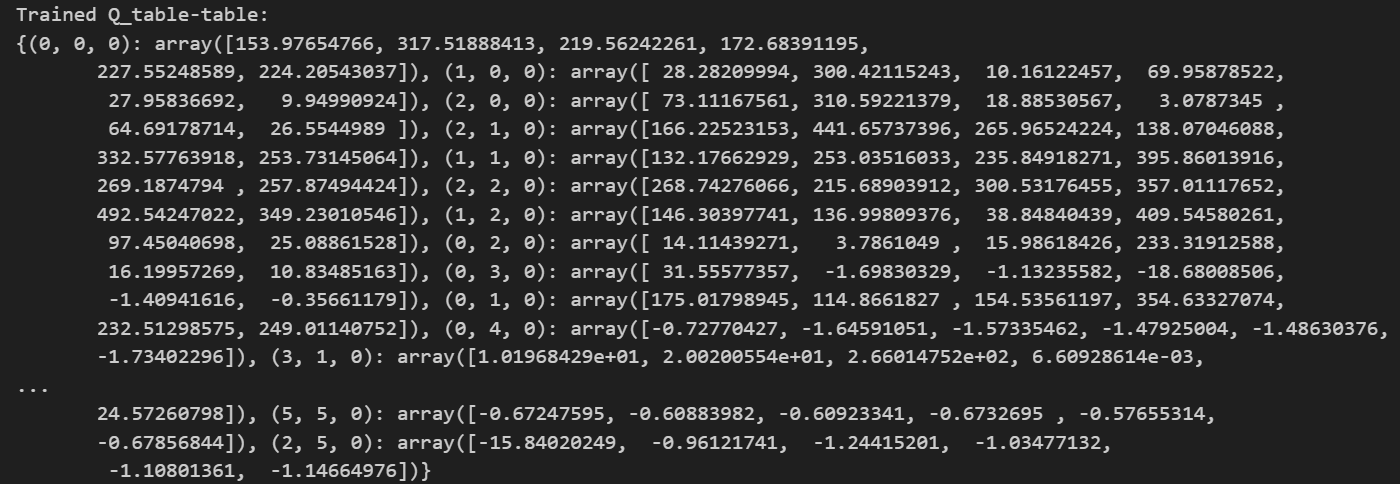
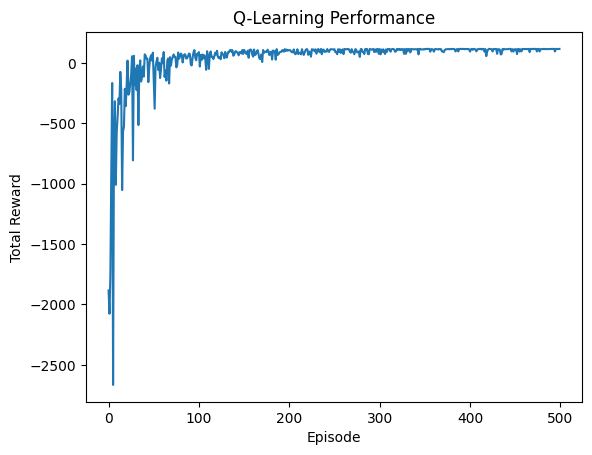
Github commits snapshot:



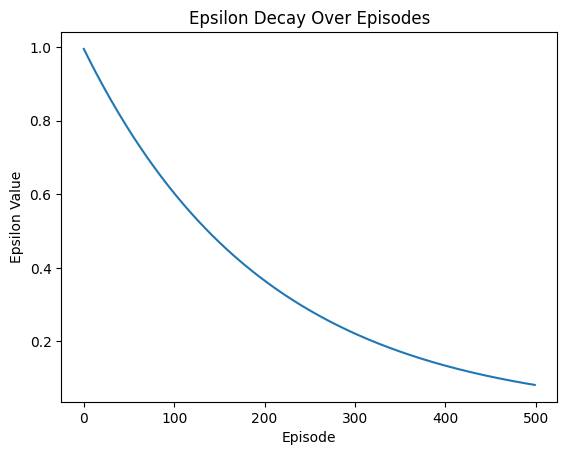
**PART 2:**

**1)Show and discuss the results after**

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

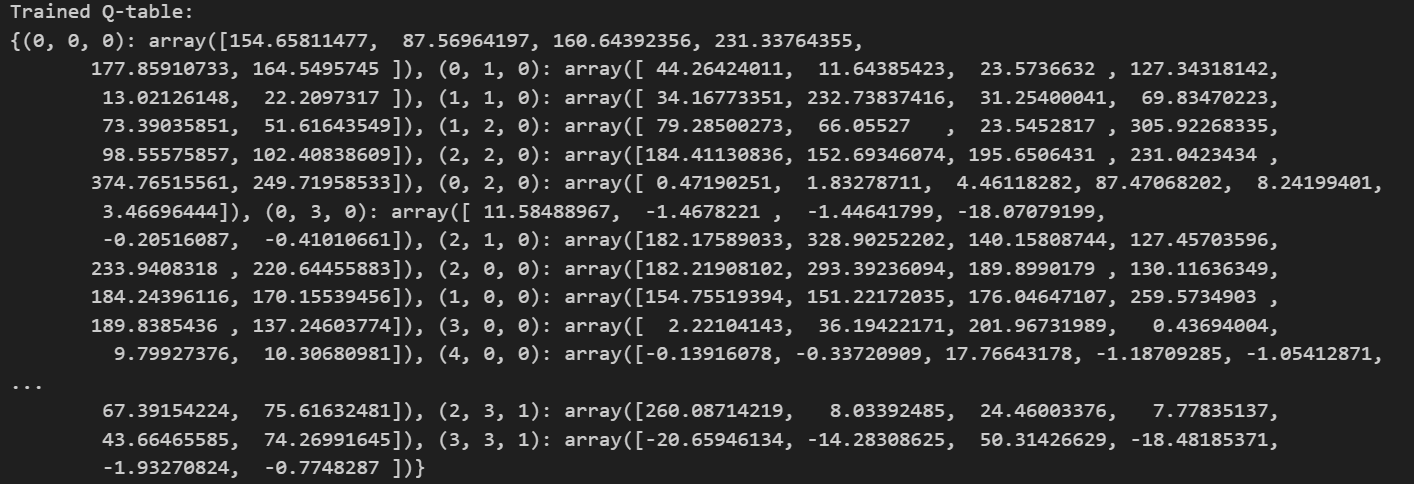
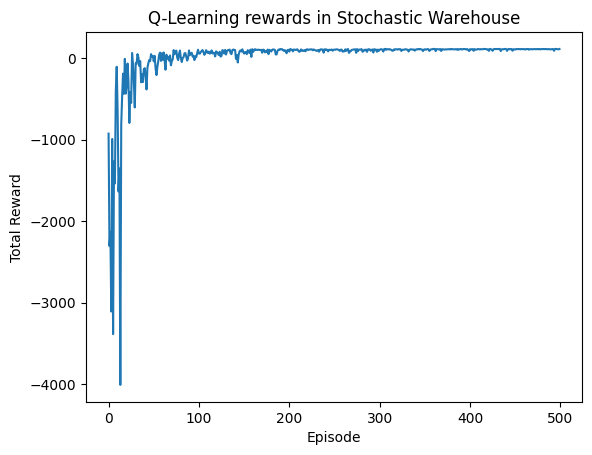
Final Q table values:  
  
Q learning performance(Total reward per episode):  


At the start of training, the robot struggles, making many random moves and getting low rewards (sometimes as bad as -2500). But as it learns from experience, it gradually improves, and the rewards start increasing. Around 100-200 episodes, the wild ups and downs settle down, showing that the robot is starting to figure things out. By 200-300 episodes, the rewards become more stable, meaning the robot has learned a solid strategy for picking up and dropping off items efficiently. The overall pattern shows that the robot starts clueless but eventually becomes much smarter through trial and error.

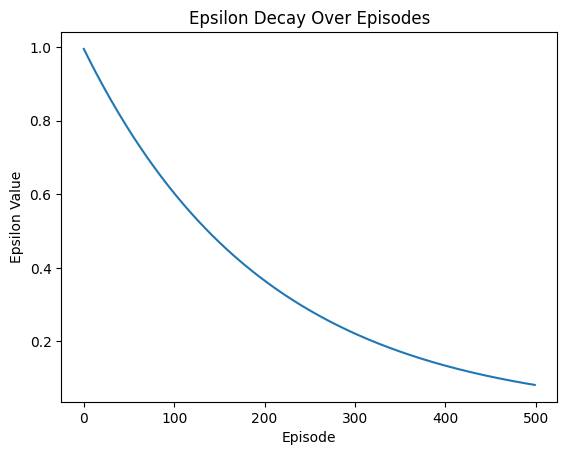
Epsilon decay:  


At the beginning of training, the robot explores completely randomly because epsilon starts at 1.0. As training goes on, epsilon gradually decreases, meaning the robot relies less on random moves and more on what it has learned. This shift happens slowly, so the robot still explores new possibilities early on but starts making smarter choices as it improves. By the later episodes, epsilon is very low, meaning the robot mostly follows its learned strategy instead of guessing. This balance between exploring and using what it knows helps the robot find the best way to complete its tasks without getting stuck in bad habits too soon.

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

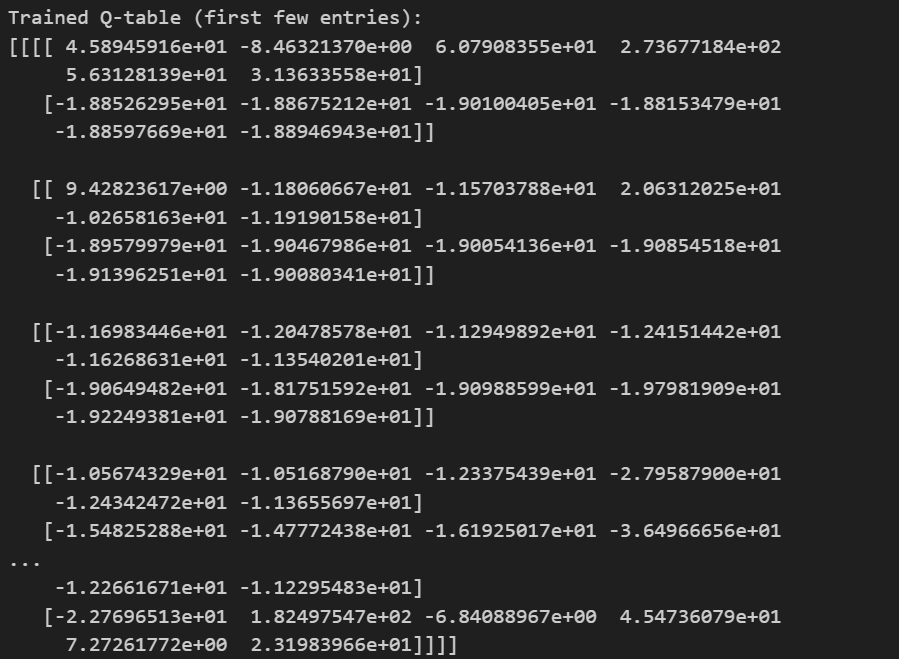
Final Q-table values:  
Q learning performance(Total reward per episode):  


At first, the robot’s performance is all over the place, with low and unpredictable rewards because it’s still figuring things out. In the first 50 episodes, it starts improving quickly as it learns better strategies. By around 100 episodes, things begin to settle, and after 200 episodes, the rewards stay mostly consistent, meaning the robot has found a solid approach. Even though the environment is random, Q-learning helps the robot adapt, proving that it can handle uncertainty well. The steady rewards at the end show that the robot has learned a reliable and effective way to complete its tasks.

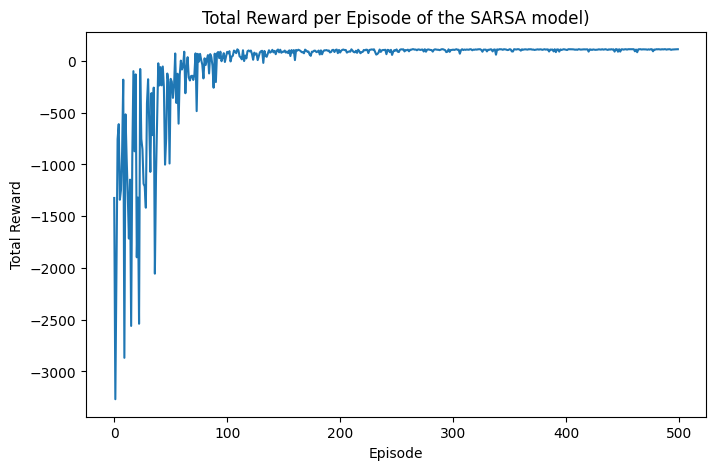
Epsilon decay:  


At the start of training, epsilon is set to 1.0, meaning the robot explores a lot and tries out different actions. As training progresses, epsilon gradually decreases, shifting the focus from exploration to using what it has learned. This helps the robot make smarter decisions based on experience rather than random guessing. By the time it reaches 500 episodes, epsilon is near zero, meaning the robot mostly follows its learned strategy. This gradual decay ensures a good balance—allowing the robot to explore enough early on while making sure it doesn’t get stuck with a bad strategy too soon.

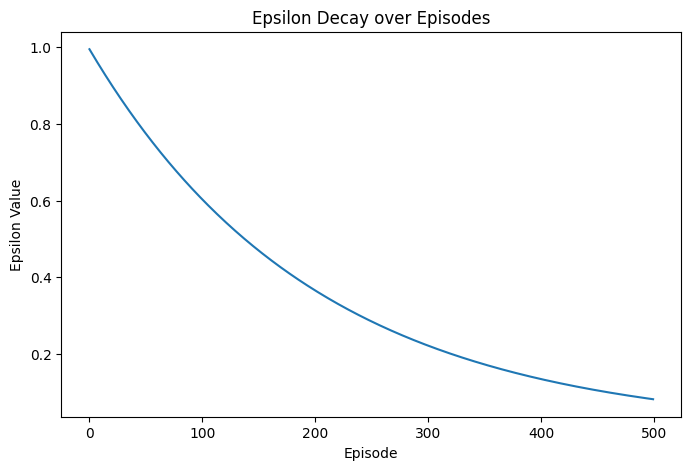
**c) Applying any other algorithm of your choice to solve the deterministic environment defined in Part 1. Plots should include total reward per episode.**

Final Q-table values:  


SARSA performance(Total reward per episode):

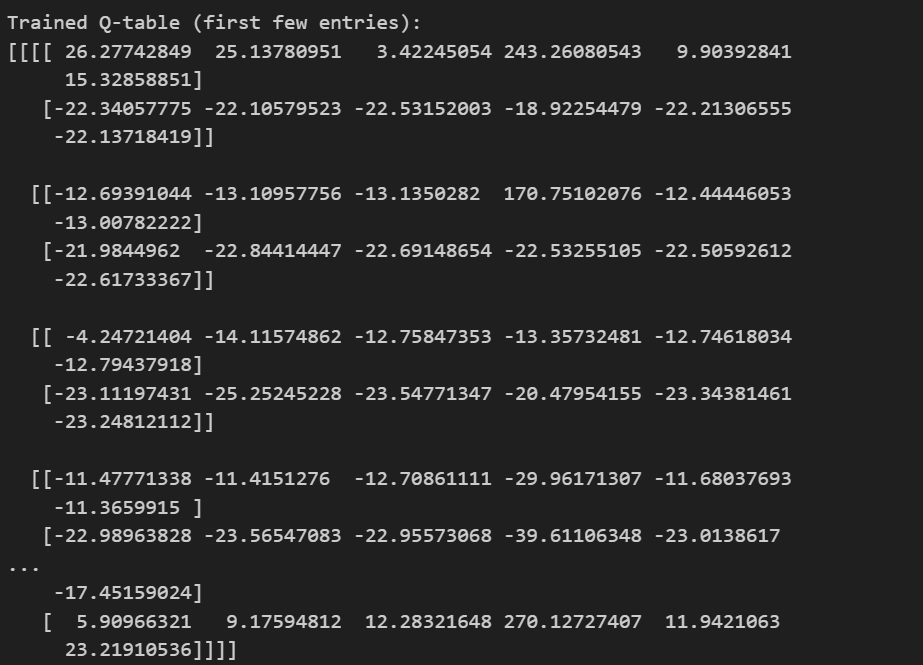


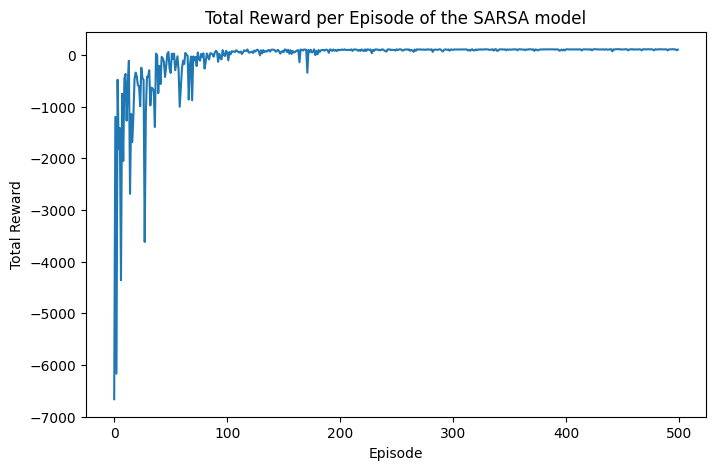
In the beginning, the robot struggles, making a lot of mistakes and receiving low rewards, which fluctuate a lot due to random exploration. As training continues, it gradually improves, with rewards steadily increasing. Around episode 100, the learning process becomes more stable, showing that the robot is figuring out a good strategy. By episode 200, the rewards level off near zero or slightly positive, meaning the robot has found a reliable way to complete its tasks. This pattern shows how SARSA effectively balances trial-and-error learning with smarter decision-making, leading to steady and consistent performance over time.

Epsilon decay:  


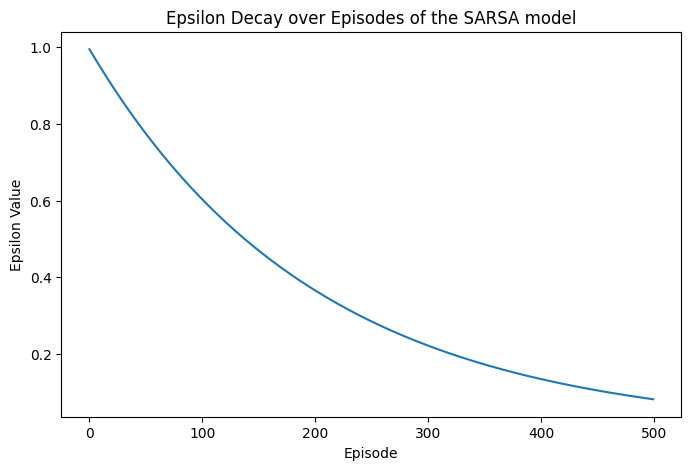
It shows how epsilon decreases over 500 episodes, starting at 1 and gradually approaching zero. This represents the balance between exploration and exploitation in reinforcement learning. In the beginning, the agent explores new actions frequently, but as training progresses, it relies more on what it has learned. The smooth, gradual decline suggests a well-planned decay strategy, ensuring the agent gathers enough diverse experiences early on while allowing it to focus on fine-tuning its policy later. This approach helps improve learning efficiency and ensures better long-term performance.

**d) Applying any other algorithm of your choice to solve the stochastic environment defined in Part 1. Plots should include total reward per episode.**

Final Q-table values:  


SARSA performance(Total reward per episode):  


It shows the total reward per episode for a SARSA model, illustrating its learning process. In the beginning, the rewards fluctuate a lot and are mostly negative, as the model explores different actions. Around 100 episodes in, the rewards start to stabilize near zero, though occasional dips show that learning is still happening. By episode 200, the model has mostly settled, maintaining steady performance with minimal variation, indicating that it has learned a stable policy.

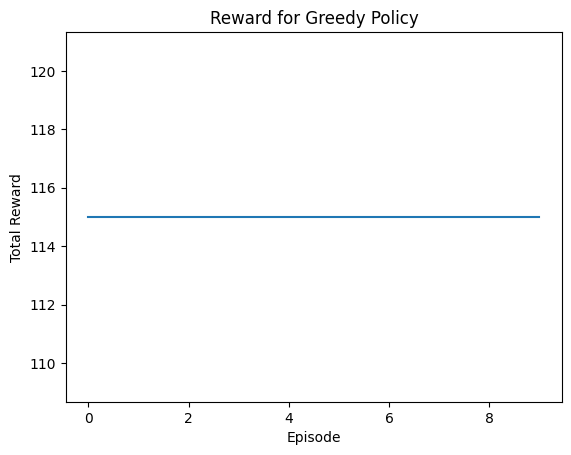
Epsilon decay:  


It shows epsilon decay in the SARSA model, showing how the exploration rate decreases over time. At the start, epsilon is 1, encouraging the agent to explore widely. As training progresses, it gradually decreases, shifting the focus from exploration to exploiting learned policies. By around 500 episodes, epsilon is very low, meaning the agent mostly relies on its learned strategy rather than taking random actions. This decay strategy helps balance exploration and exploitation, allowing the agent to learn effectively in the early stages while ensuring stable policy convergence later.

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

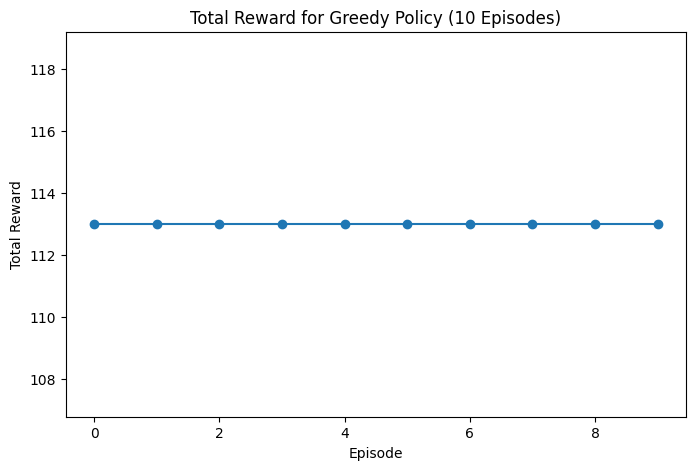
Deterministic environment:

1. Q Learning:



It shows the total reward over episodes when using a greedy policy, where the agent always chooses the action with the highest Q-value. The flat line indicates that the total reward stays the same across episodes, suggesting a stable and consistent policy. This means the agent has fully learned an optimal strategy and reliably achieves the same reward in every episode.

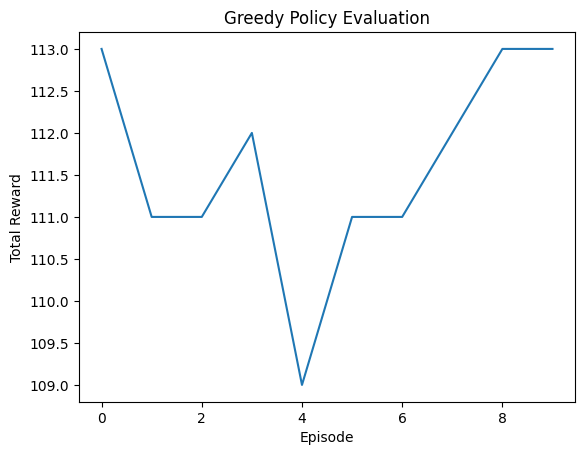
SARSA:



It displays the total reward per episode for a greedy policy over 10 episodes, consistently hovering around 113. This indicates that the policy has fully stabilized, delivering the same performance in each episode. Since a greedy policy always picks the best-known action, it suggests that the agent has learned an optimal or near-optimal strategy. The lack of fluctuations confirms that exploration has stopped, and the model is now purely exploiting its learned knowledge.

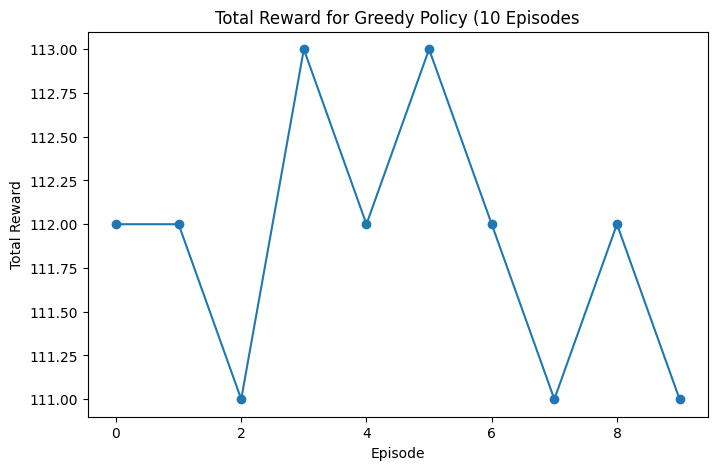
Stochastic Environment:

1. Q Learning:



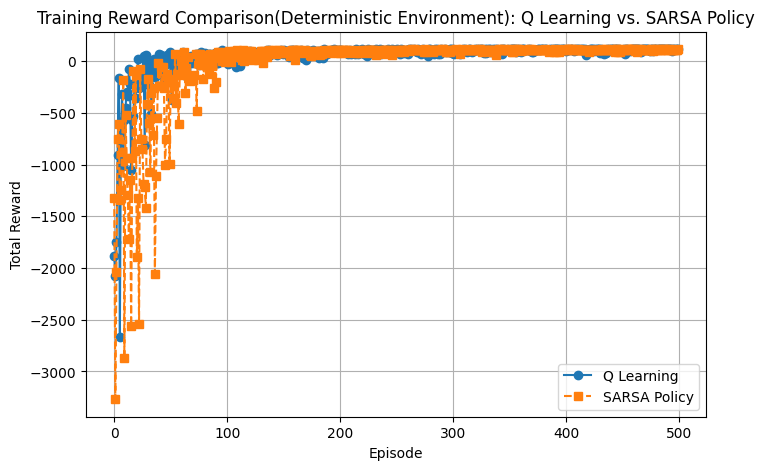
It evaluates a greedy policy by showing total rewards across episodes. The fluctuations suggest some variation in performance, likely due to the environment’s randomness. However, the consistently high rewards indicate that the learned policy is effective. The slight instability may stem from residual exploration, environmental randomness, or occasional suboptimal choices. Still, the upward trend toward the end suggests that the policy stabilizes and achieves reliable performance over time.

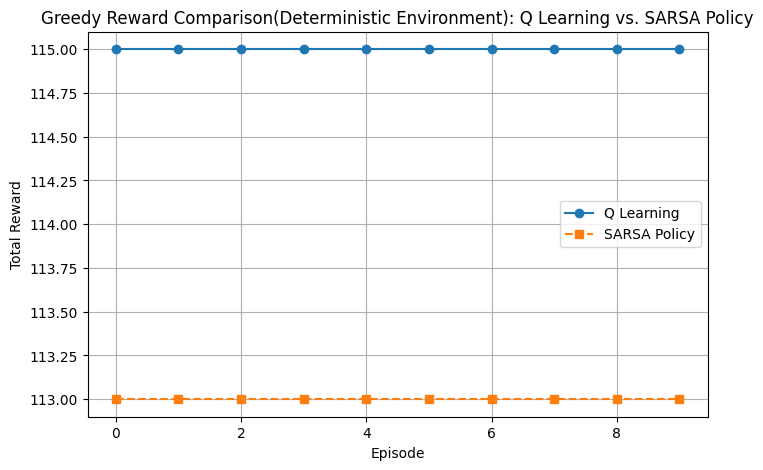
SARSA:



It depicts the total reward over 10 episodes using a greedy policy, where the agent strictly follows its learned strategy without exploration. The rewards fluctuate slightly within a narrow range, indicating stable but variable performance. The peaks and dips suggest sensitivity to environmental randomness or variations in state-action transitions, but overall, the policy remains consistent in achieving high rewards.

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

Total reward per episode:  


Greedy reward comparison:  


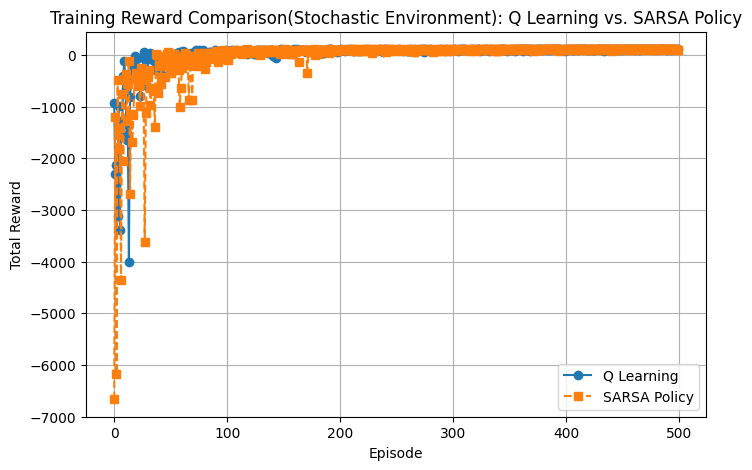
Interpretation:  
Training Reward Comparison:

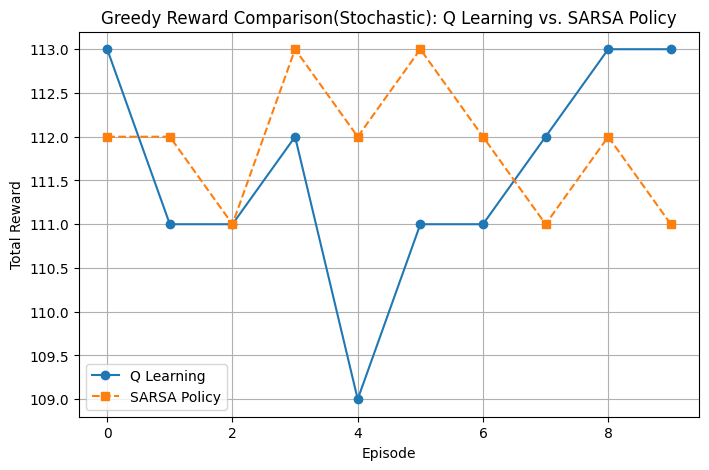
At the start, both Q-Learning and SARSA experience significant negative rewards as they explore the environment. Over time, they refine their policies and move toward higher rewards. Q-Learning shows more variation early on due to its off-policy nature, while SARSA, being on-policy, progresses more smoothly. Eventually, both algorithms stabilize at a similar optimal reward level, but Q-Learning reaches it more efficiently.

Greedy Reward Comparison:

Q-Learning achieves a slightly higher total reward (~115) compared to SARSA (~113), indicating a marginally better final performance. Both algorithms have converged to their optimal policies, but Q-Learning's maximization strategy gives it an edge. In a deterministic environment, this allows Q-Learning to achieve higher rewards by always selecting the best possible action.

3) Compare how both algorithms perform in the same stochastic environment (e.g. show one graph with two reward dynamics) and give your interpretation of the results.

Total reward per episode:  


Greedy reward comparison:  


Interpretation:  
Training Reward Comparison:

The graph shows that both Q-Learning and SARSA begin with significantly negative rewards but improve over time. Early on, SARSA experiences more extreme negative rewards, suggesting greater sensitivity to exploration in the stochastic environment. As training progresses, both algorithms converge toward higher rewards, indicating successful learning. SARSA stabilizes more quickly, while Q-Learning exhibits slightly more fluctuation in the early episodes.

Greedy Reward Comparison:

Both algorithms reach similar final rewards, hovering around 110-113. However, SARSA shows a more stable reward pattern, while Q-Learning has more noticeable ups and downs. This fluctuation in Q-Learning is expected since it takes a more aggressive, off-policy approach, whereas SARSA’s on-policy updates lead to smoother learning.

**4) Briefly explain the tabular methods, including Q-learning, that were used to solve the problems. Provide their update functions and key features.**

Q Learning:

**Key features:**  
Off-policy learning: Q-learning updates its Q-values based on the highest possible future reward, rather than strictly following its current policy.

Exploration-exploitation tradeoff: Uses an epsilon-greedy policy to balance exploration and exploitation.

Converges to the optimal policy if given sufficient exploration and training episodes.

**Update function:**

Where:

* Q(s,a) is the current Q-value for state s and action a.
* α (learning rate) determines how much new information overrides old values.
* r is the reward received after taking action a in state s.
* γ\gamma (discount factor) determines the importance of future rewards.
* maxa′Q(s′,a′) represents the best possible action in the next state s′.

**Implementation Details:**

a) The agent selects an action using an epsilon-greedy policy.

b) After taking action, the best next action is chosen greedily (maxQ).

c) Q-values are updated using the max future value.

**Advantage:**

* Learns an optimal policy even if the policy being followed during training is different.

**Disadvantage:**

* High variance due to selecting the maximum estimated future reward, which may not always be accurate in the early training stages.

SARSA:

**Key features:**  
On-policy learning: SARSA updates its Q-values using the next action selected by its current policy, rather than the maximum future action like Q-learning.

More stable than Q-learning since it follows the current policy, avoiding high-variance updates.

Converges to a policy that considers exploration, leading to safer strategies in high-risk environments.

**Update function:**  
Where:

* a′ is the next action chosen by the **current policy** (instead of taking the max Q-value like in Q-learning).
* α (learning rate) determines how much new information overrides old values.
* r is the reward received after taking action a in state s.
* γ\gamma (discount factor) determines the importance of future rewards.

**Implementation Details:**

a)The agent selects an action using the epsilon-greedy policy.

b)Instead of choosing the best next action greedily, it follows the same epsilon-greedy policy to choose the next action.

c)Q-values are updated using the next action’s Q-value (not necessarily the max value).

Advantage:

* More stable learning in environments where exploration is necessary.
* Safer policies in stochastic environments because it does not always assume the best possible outcome (unlike Q-learning).

Disadvantage:

* May converge to a suboptimal policy if not enough exploration is done.

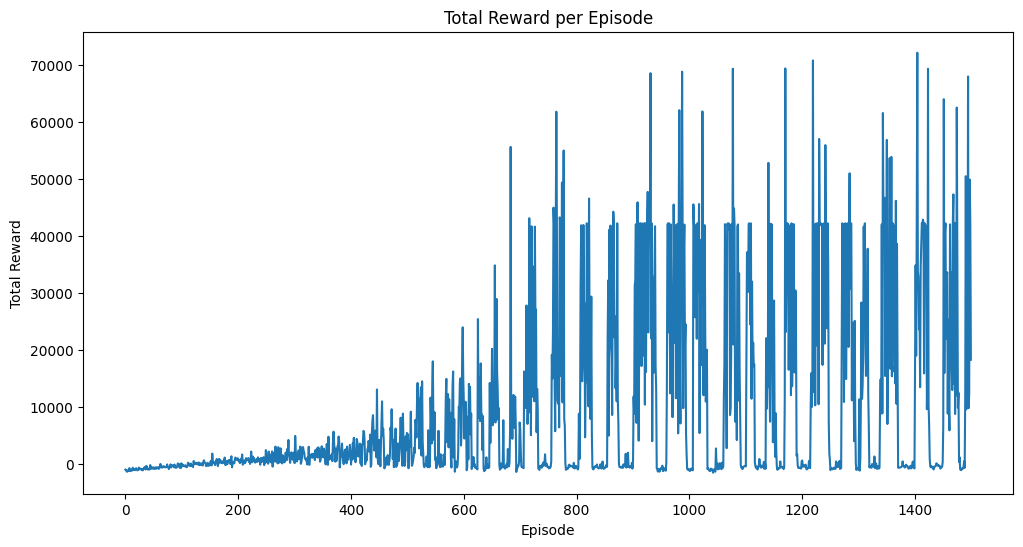
**5) Briefly explain the criteria for a good reward function. If you tried multiple reward functions, give your interpretation of the results.**

A well-designed reward function is crucial for guiding reinforcement learning agents towards optimal behavior. The following are key criteria for a good reward function:

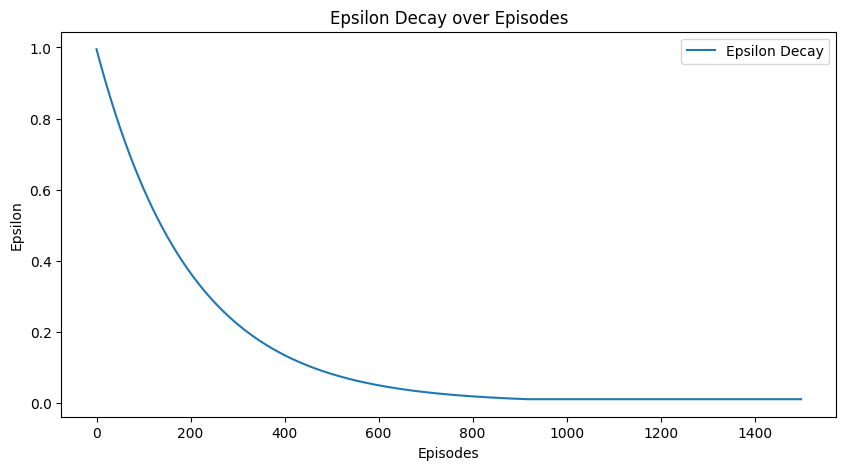
1. Clear and Aligned with the Goal – The reward should directly reflect the desired outcome, ensuring that maximizing it leads to the correct behavior.
2. Encourages Learning Efficiency – It should provide enough feedback to guide learning effectively, avoiding sparse or excessively delayed rewards.
3. Balances Short-Term and Long-Term Goals – A well-designed function considers both immediate and future rewards, using an appropriate discount factor (γ).
4. Avoids Unintended Exploits – Poorly designed rewards can lead to unintended behaviors where the agent finds loopholes to maximize rewards in unintended ways.
5. Scalability and Stability – Reward magnitudes should be appropriately scaled to prevent instability. Extremely large or small rewards can slow learning or cause erratic updates.
6. Encourages Exploration – If rewards are too predictable, the agent may settle for suboptimal strategies instead of discovering better ones.

**PART 3**

**1)Show and discuss the results after applying the Q-learning algorithm to solve the stock trading problem. Plots should include epsilon decay and total reward per episode.**

Total reward per episode:  


In the early phase (0-400 episodes), rewards gradually increase, indicating that the agent is learning and improving its policy. During the mid-phase (400-800 episodes), variance rises as the agent explores different strategies, though the overall trend remains positive. In the later phase (800+ episodes), sharp fluctuations occur, with sudden spikes and drops suggesting instability, possibly due to aggressive exploration, environmental changes, or abrupt model updates.

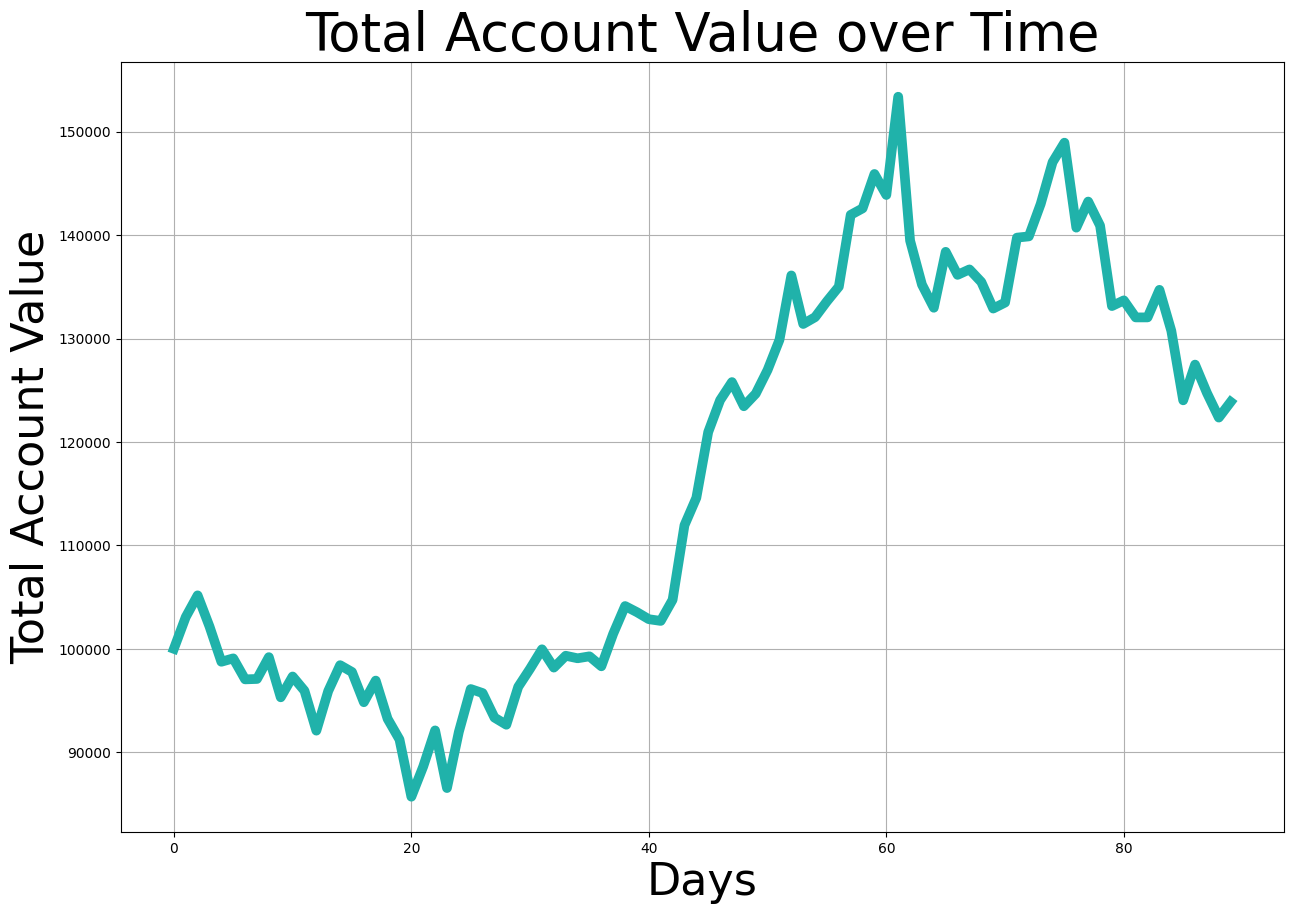
Epsilon decay:  


The epsilon decay curve starts at 1.0, encouraging high exploration, and gradually decreases to promote the exploitation of learned policies. The steep initial drop signifies a rapid reduction in random actions, ensuring sufficient early exploration, while the slower decay after 600-800 episodes marks a transition to more stable decision-making. The final epsilon value remains slightly above zero, maintaining minimal exploration to avoid getting stuck in local optima. This strategy effectively balances exploration and exploitation, but careful tuning of the decay rate is essential—dropping too quickly can lead to suboptimal learning, while a slow decay may unnecessarily extend training.

**2) Provide the evaluation results. Evaluate your trained agent’s performance (you will have to set the train parameter set to False), by only choosing greedy actions from the learnt policy. Plot should include the agent’s account value over time. Code for generating this plot is provided in the environment’s render method. Just call environment.render after termination**

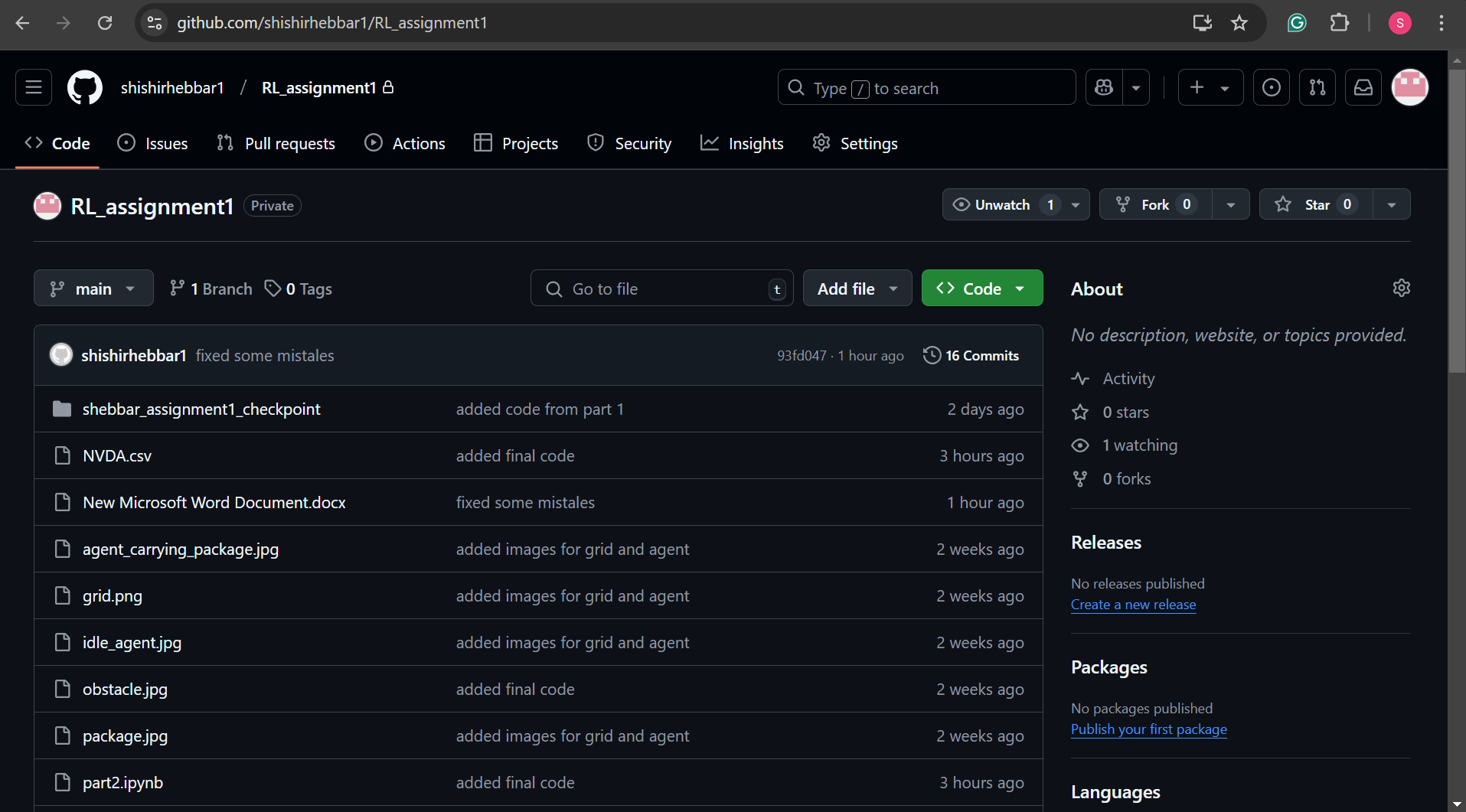
Evaluation rewards: 1463.7479220411012

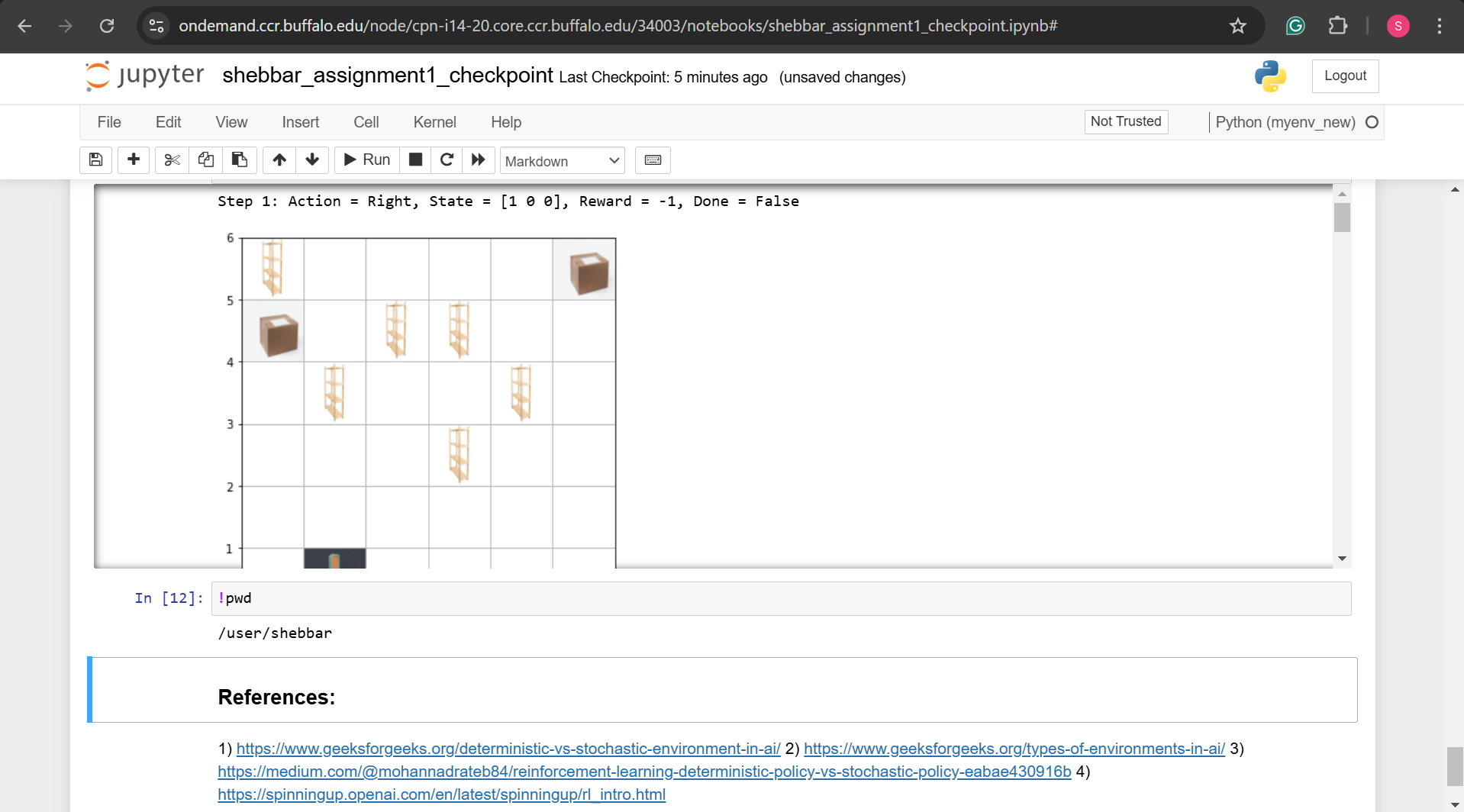
Total account value with time:

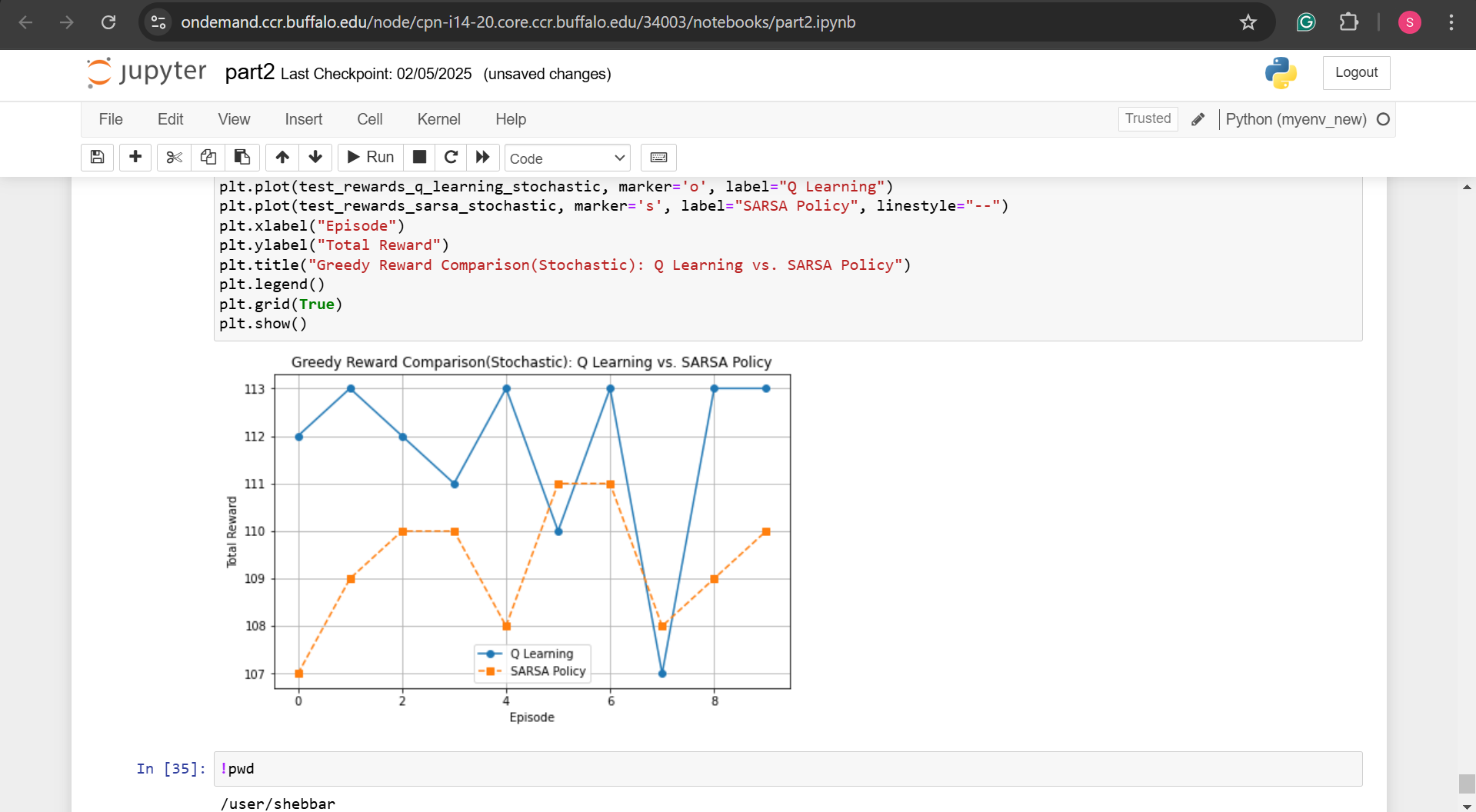


The evaluation results highlight the trained agent’s performance in the financial trading environment, where it follows a greedy policy by selecting the best actions based on its learned strategy. Initially, the total account value experiences a drawdown, likely due to early suboptimal trades or market volatility, but it soon enters a strong growth phase, showcasing the agent’s ability to identify and capitalize on profitable opportunities. The portfolio peaks around days 60-70 at approximately 150,000 before experiencing some declines, likely caused by market fluctuations. Despite this, the final account value remains significantly higher than the starting point, demonstrating overall profitability.

**Final Git Bonus:**



CCR Bonus:  
PART 1:  


PART 2:  
  
Part 3:

