

Assignment1 - Defining and Solving RL Environments
Aniket Kumar - 50608890
akumar74@buffalo.edu

1. Defining RL environments

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

Deterministic states consist of 6x6 states.

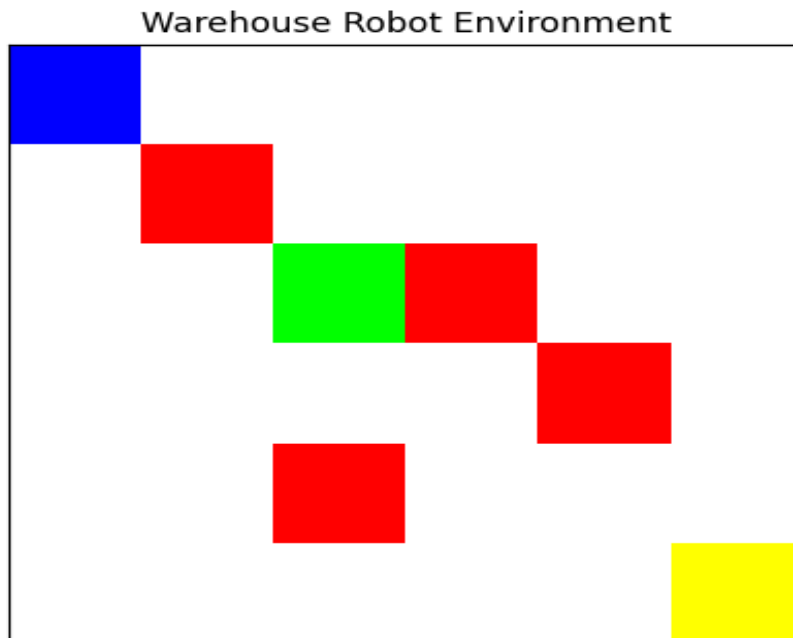
The actions consist of 6, down, up, left, right, pickup and dropoff.

The rewards also consist of 4.

The rewards are +100pt for successful delivery, +25pts for picking up the object for the first time, -1pt for each step taken to encourage efficiency, -25pts if the dropoff location is wrong and -20pts for hitting the obstacles based on the action.

Stochastic environments are based on deterministic environments. So the states, actions, and rewards are the same with the deterministic environment. The only difference is that action is decided based on probability. The objective of two environments (deterministic, stochastic) are finding optimal policy which maximizes the return (rewards).

2. Provide visualizations of your environments



Blue Box - position of robot (0,0)

Red Box - position of obstacles [(1,1),(2,3),(4,3),(4,2)]

Green Box - pickup location (2,2)

Yellow Box - dropoff location (5,5)

3. How did you define the stochastic environment?

The mentioned difference of deterministic and stochastic environment was the action of the stochastic environment is decided based on probability. The actions 0,1,2,3,4,5 mean down, up, right, and left, pickup and dropoff each. If action is 0(down), then 55% probabilities will be given to action 'down', and the rest will have 15% probabilities each.

4. What is the difference between the deterministic and stochastic environments?

In deterministic policy, the action must be mapping by the current state. The formula is $a = \pi(s)$.

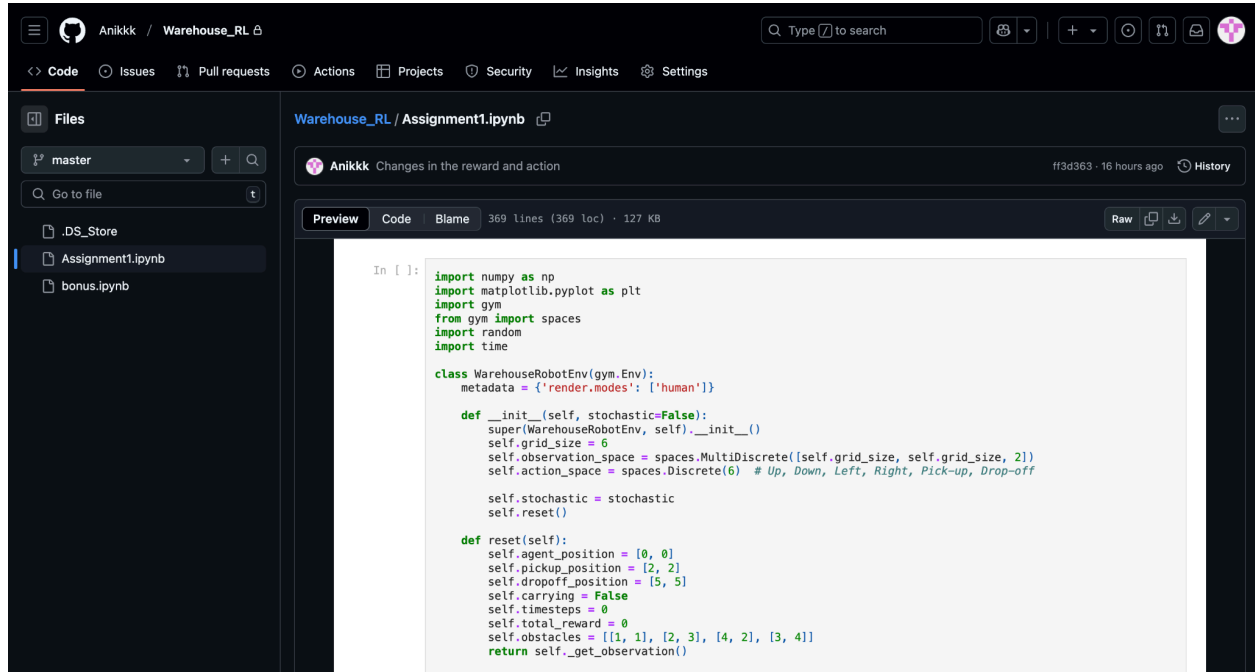
In stochastic policy, the action will be selected stochastically by the current state. The formula is $a = \pi(a|s)$. The added values of probabilities are 1.

5. Safety in AI

I ensure the safety of my environment by using the numpy clip method. If I specify the minimum and maximum values in the argument, I could prevent out-of-range values. I assigned minimum 0 and maximum 5 values, so if the number is out of the range, I made the number would be replaced with those numbers. This is the more specific explanation why we should use the clip method or checking the boundary condition. If the agent is in the [5,5] position, the agent cannot go down or right. So by specifying the maximum value to 5, we could replace out-of-range numbers to 5. On the other hand, if the agent is in the [0,0] position, the agent cannot go up or left. So we could replace out-of-range numbers to 0 by specifying the minimum value to 0.

Bonus Task:

Git Expert



The screenshot shows a GitHub repository named 'Warehouse_RL' by user 'Anikkk'. The file 'Assignment1.ipynb' is selected, showing its commit history and code. The code is a Python class 'WarehouseRobotEnv' that inherits from 'gym.Env'. It defines the environment's metadata, initialization, reset, and observation methods. The environment is a 6x6 grid with a robot at [0, 0], a pickup location at [2, 2], and a dropoff location at [5, 5]. Obstacles are located at [1, 1], [2, 3], [4, 2], and [3, 4].

```
In [ ]: import numpy as np
import matplotlib.pyplot as plt
import gym
from gym import spaces
import random
import time

class WarehouseRobotEnv(gym.Env):
    metadata = {'render.modes': ['human']}

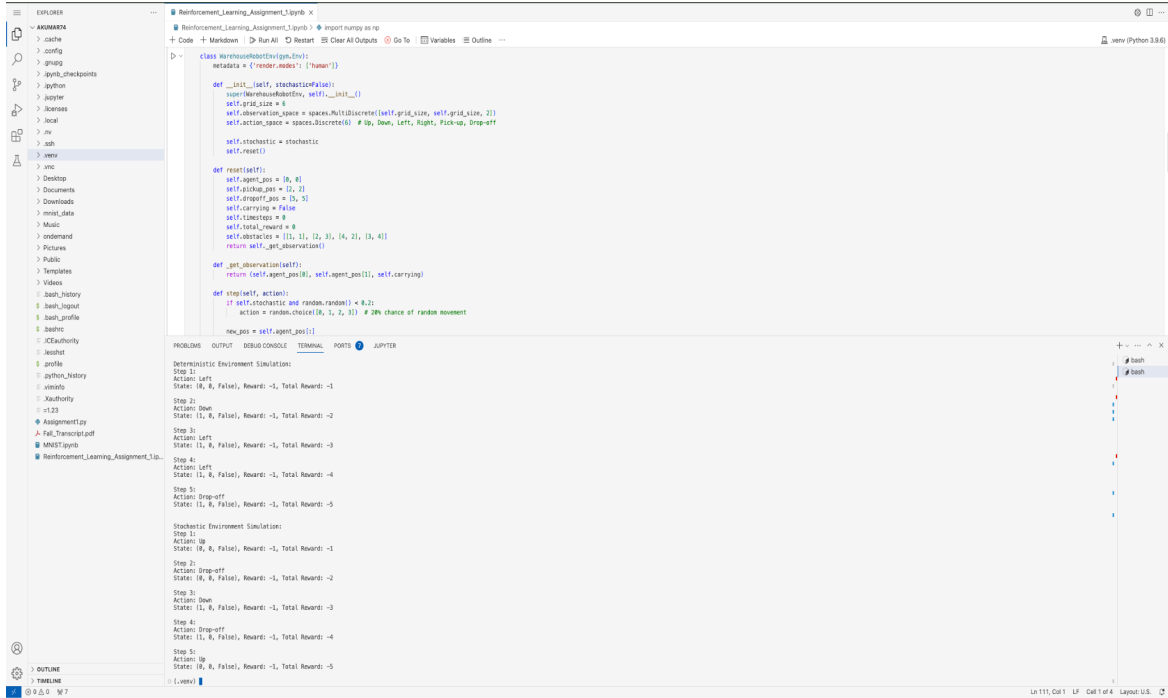
    def __init__(self, stochastic=False):
        super(WarehouseRobotEnv, self).__init__()
        self.grid_size = 6
        self.observation_space = spaces.MultiDiscrete([self.grid_size, self.grid_size, 2])
        self.action_space = spaces.Discrete(6) # Up, Down, Left, Right, Pick-up, Drop-off

        self.stochastic = stochastic
        self.reset()

    def reset(self):
        self.agent_position = [0, 0]
        self.pickup_position = [2, 2]
        self.dropoff_position = [5, 5]
        self.carrying = False
        self.timesteps = 0
        self.total_reward = 0
        self.obstacles = [[1, 1], [2, 3], [4, 2], [3, 4]]
        return self._get_observation()
```

CCR Submission

Running the code in VS Code: The screenshot of the output below-



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS 1
Deterministic Environment Simulation:
Step 1:
Action: Left
State: (0, 0, False), Reward: -1, Total Reward: -1

Step 2:
Action: Down
State: (1, 0, False), Reward: -1, Total Reward: -2

Step 3:
Action: Left
State: (1, 0, False), Reward: -1, Total Reward: -3

Step 4:
Action: Left
State: (1, 0, False), Reward: -1, Total Reward: -4

Step 5:
Action: Drop-off
State: (1, 0, False), Reward: -1, Total Reward: -5

Stochastic Environment Simulation:
Step 1:
Action: Up
State: (0, 0, False), Reward: -1, Total Reward: -1

Step 2:
Action: Drop-off
State: (0, 0, False), Reward: -1, Total Reward: -2

Step 3:
Action: Down
State: (1, 0, False), Reward: -1, Total Reward: -3

Step 4:
Action: Drop-off
State: (1, 0, False), Reward: -1, Total Reward: -4

Step 5:
Action: Up
State: (0, 0, False), Reward: -1, Total Reward: -5

(.venv) pwd
/user/akumar74
(.venv) █
```

Grid-World Scenario

Grid - 6*6

Blue Box - Agent position - (0*0)

Red Box - Obstacles position - (1*1),(2*3),(3*4),(4*2)

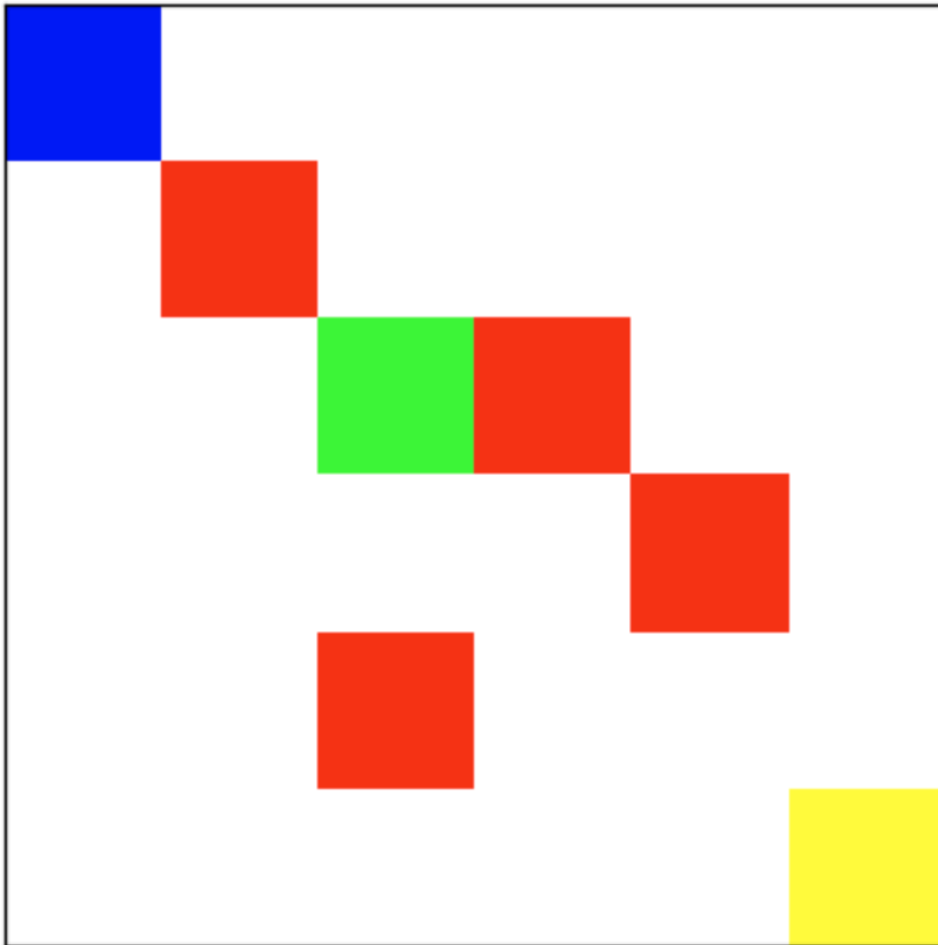
Green Box - Pickup position - (2*2)

Yellow Box - Dropoff position - (5*5)

Deterministic Environment Simulation:

Step 1:

Warehouse Robot Environment: Stochastic=False

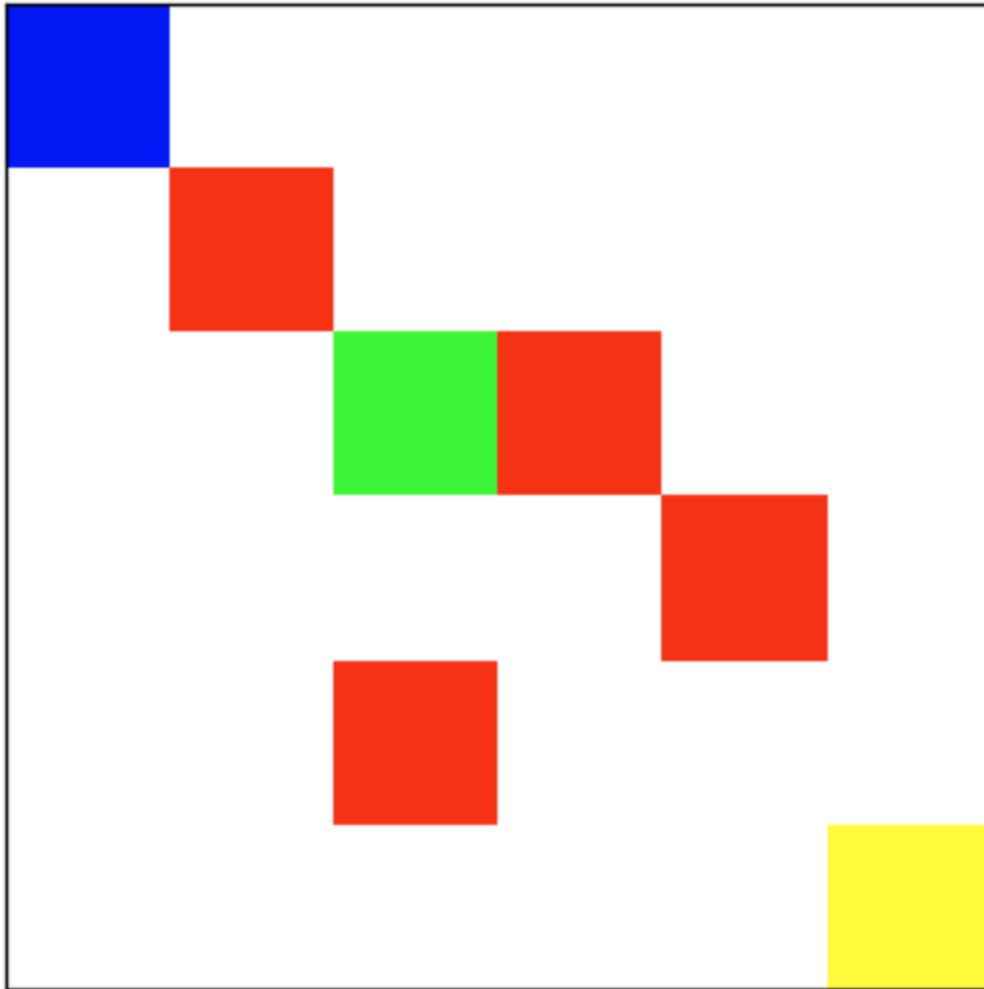


Action: Pick-up

State: (0, 0, False), Reward: -1, Total Reward: -1

Stochastic Environment Simulation:
Step 1:

Warehouse Robot Environment: Stochastic=True



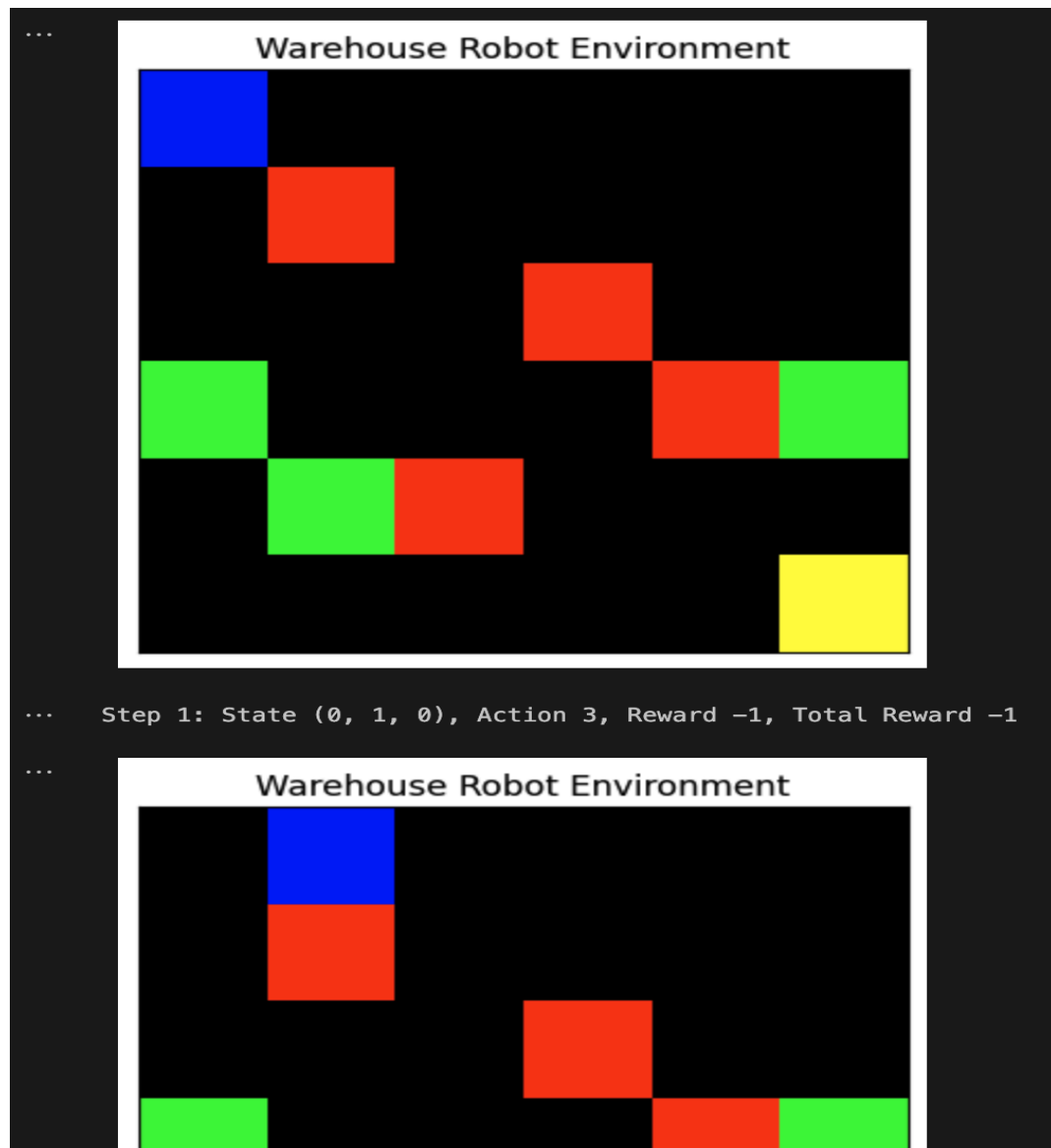
Action: Drop-off

State: (0, 0, False), Reward: -1, Total Reward: -1

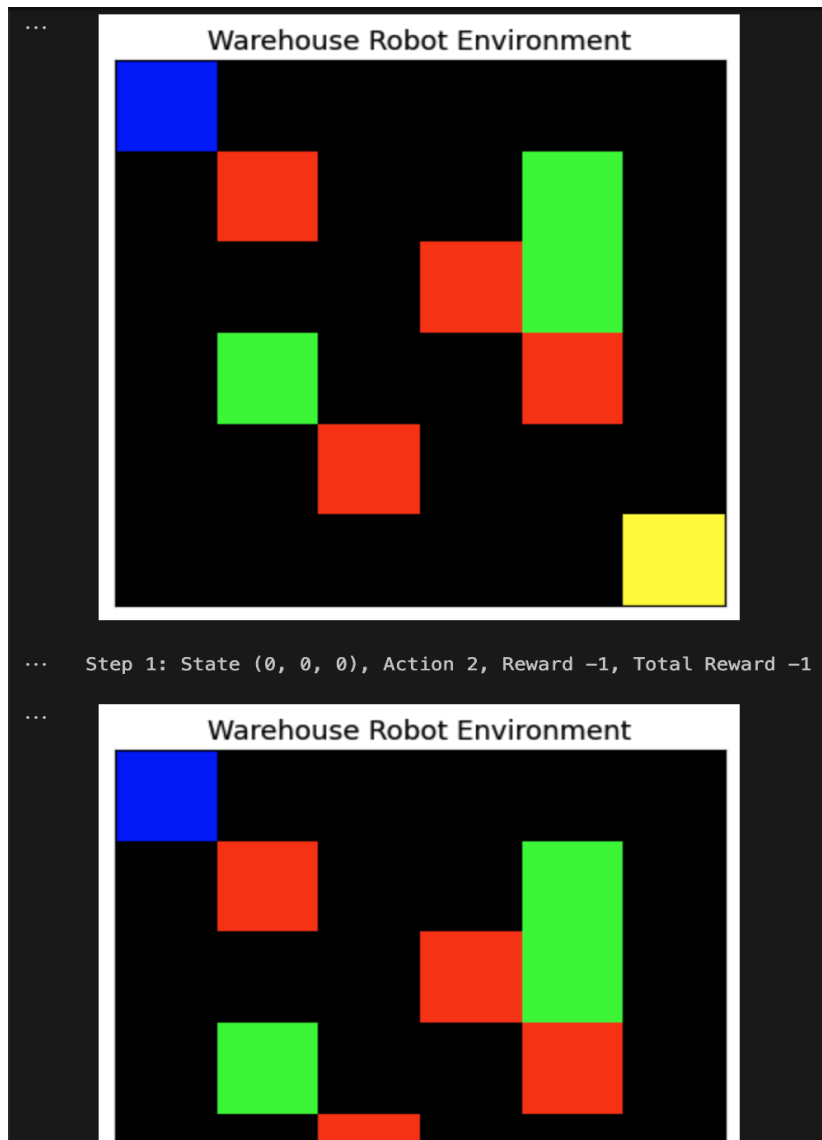
Complex Environment Scenario

The robot should be capable of multiple pick-up and drop-off tasks in a single episode. The items have to be picked up from randomly assigned locations(Bonus Task):

The Green Box represents multiple pickup location: (Iteration 1) which runs for 5 episodes

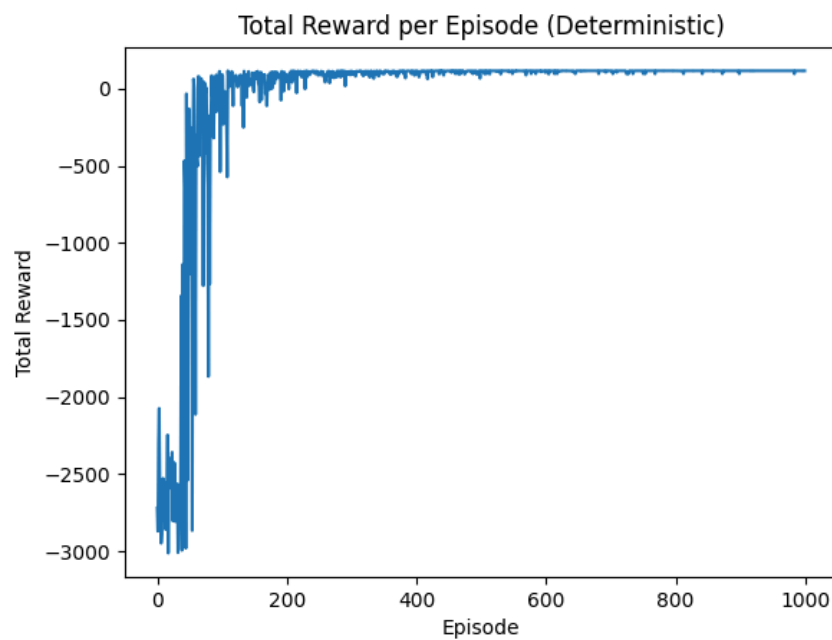
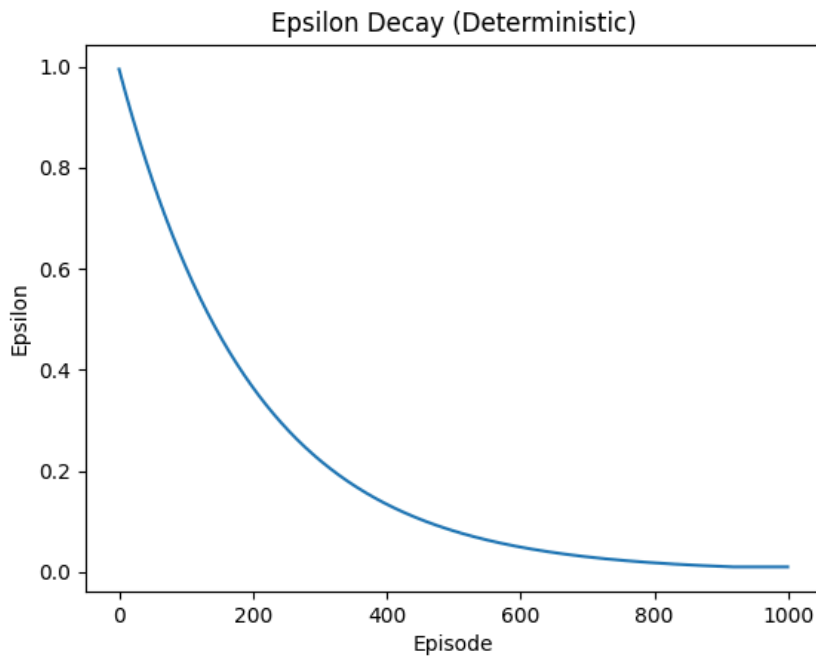


The Green Box pick location changed: (iteration 2) which again runs for 5 episodes

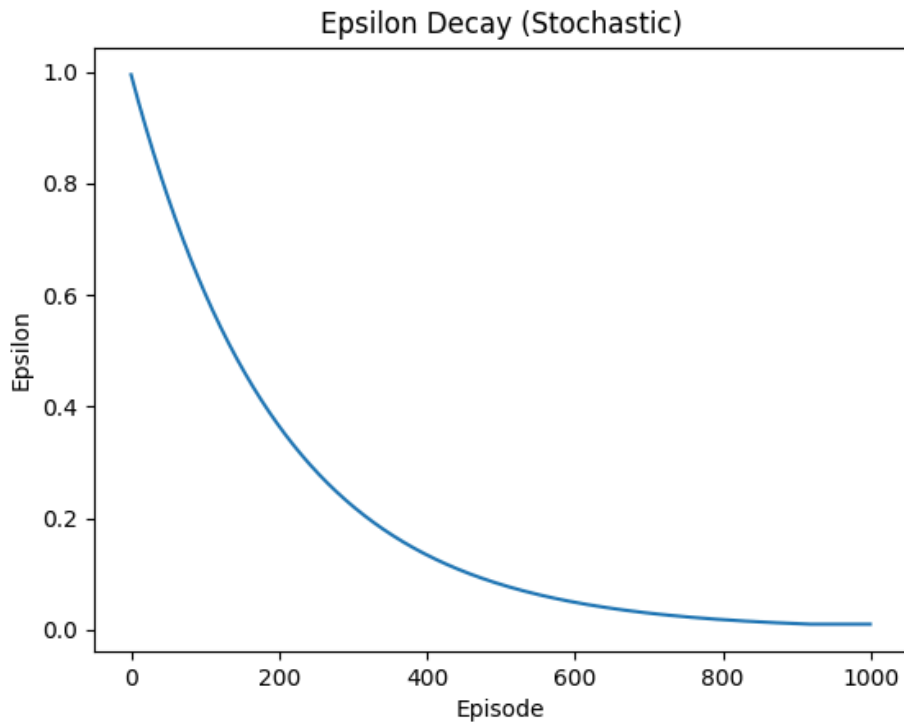


PART 2:

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

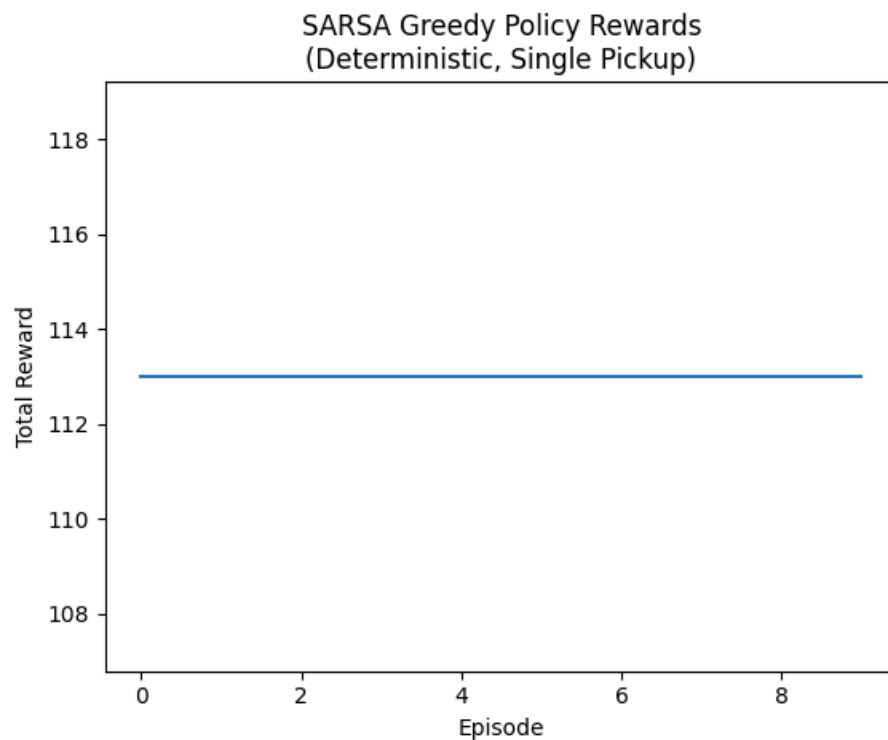
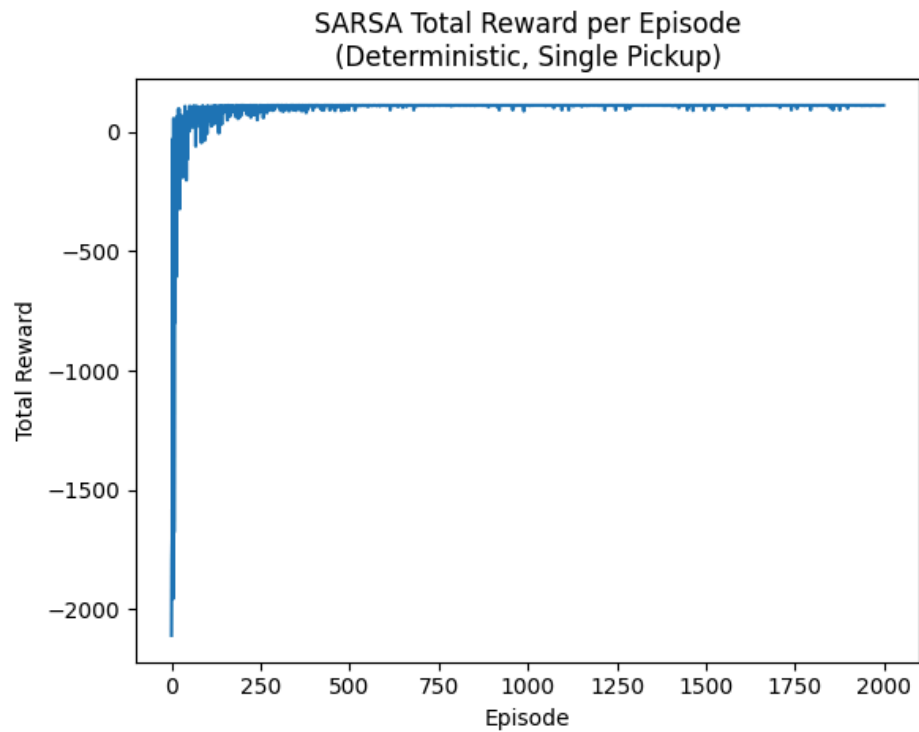


- Applying Q-learning to solve the stochastic environment defined in Part 1. Plots should include epsilon decay and total reward per episode.



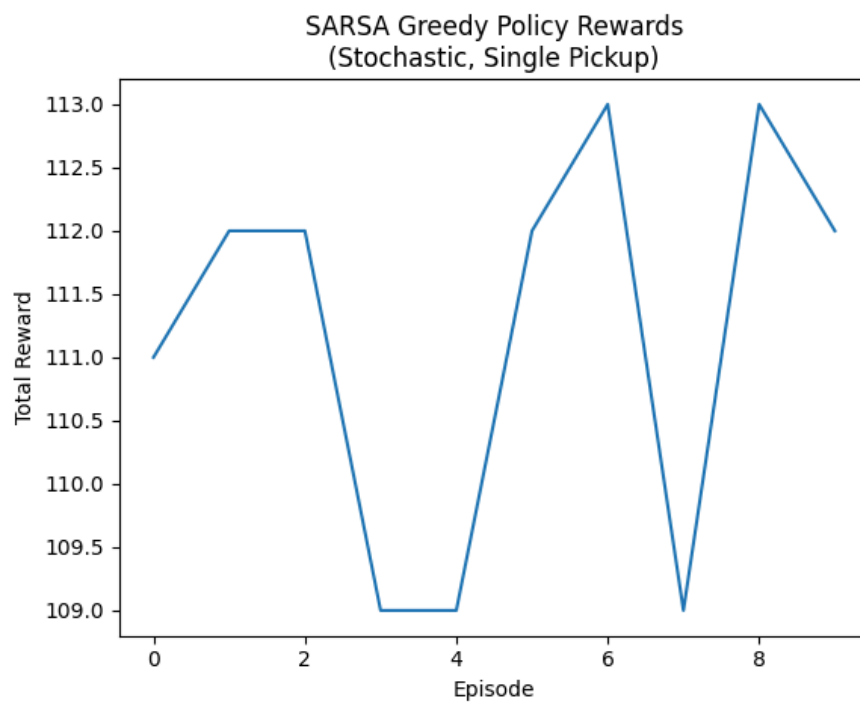
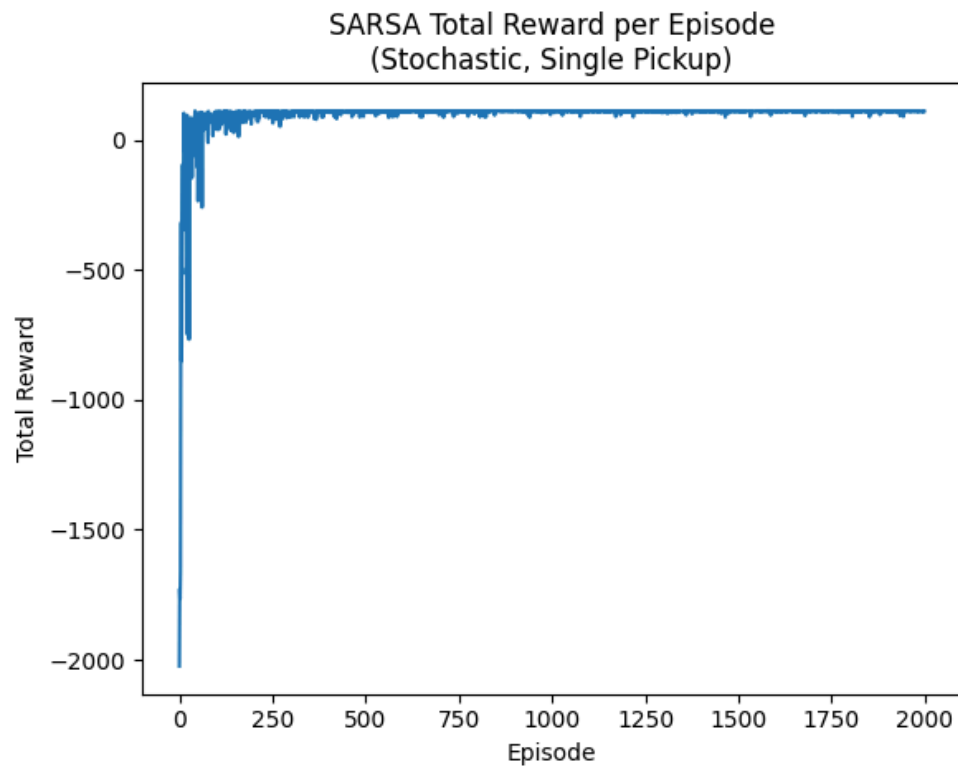
- Applying any other algorithm of your choice to solve the deterministic environment defined in Part 1. Plots should include total reward per episode.

SARSA algorithm(Deterministic Environment):



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

SARSA algorithm(Stochastic Environment):



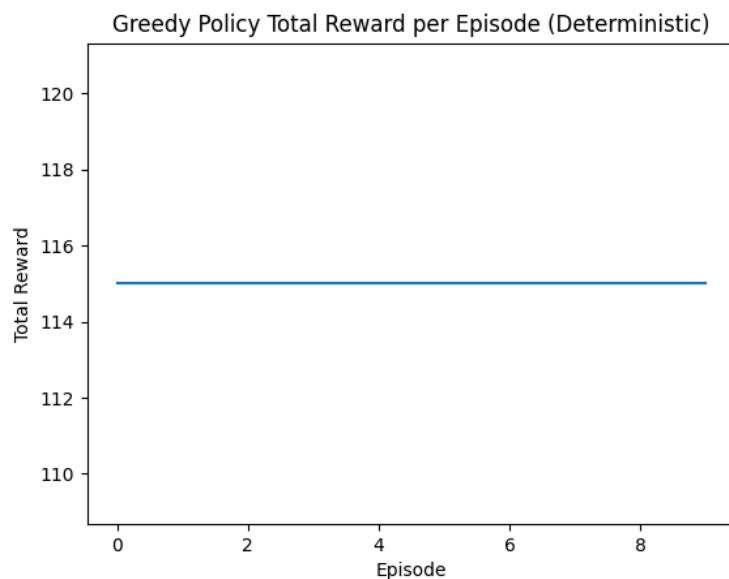
- **Provide the evaluation results. Run your environment for at least 10 episodes, where the agent chooses only greedy actions from the learnt policy. Plot should include the total reward per episode.**

The evaluation results for 10 episodes, where the agent selects only greedy actions from the learned policy, are as follows:

Q Learning:(Deterministic):

Deterministic Environment

- **Rewards:** The total reward remains constant at 115 across all 10 episodes.
- **Average Reward:** 115.0
- **Observation:** The deterministic environment ensures consistent performance, with no variation in rewards.



Q Learning:(Stochastic):

Stochastic Environment

- **Rewards:** The total reward varies across episodes:[114, 114.5, 115, 113, 111, 113.5, 112, 113, 111.5, 114]
- **Observation:** The stochastic environment introduces variability in rewards due to its inherent randomness.



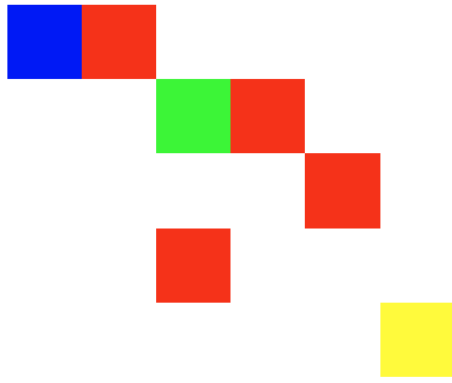
- Run your environment for 1 episode where the agent chooses only greedy actions from the learned policy. Render each step of the episode and verify that the agent has completed all the required steps to solve the environment. Save this render and include it in your report as clearly ordered screenshots or as a clearly named video file in your submission.

DETERMINISTIC ENVIRONMENT:

The screenshot below demonstrates the agent using greedy actions derived from its learned policy to reach the target position.

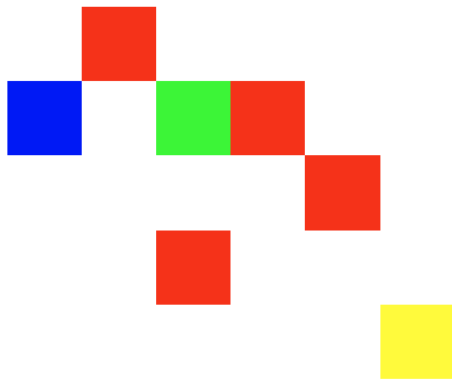
Rendering one episode with learned policy:

Warehouse Robot (Stochastic: False)



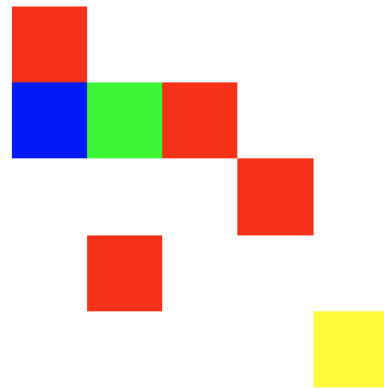
Action: Down, Reward: -1

Warehouse Robot (Stochastic: False)



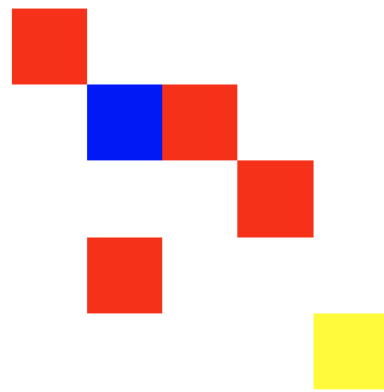
Action: Down, Reward: -1

Warehouse Robot (Stochastic: False)



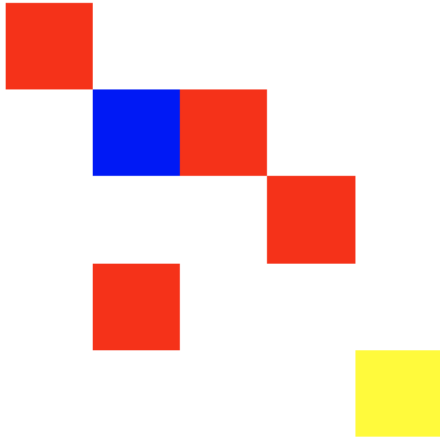
Action: Right, Reward: -1

Warehouse Robot (Stochastic: False)



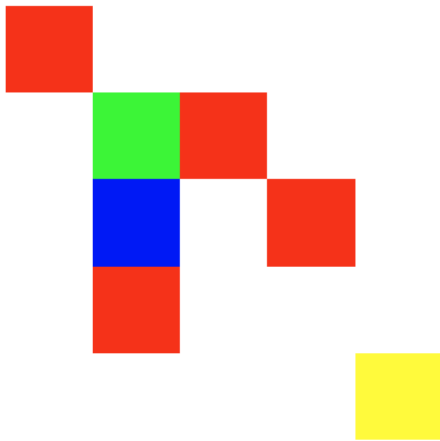
Action: Right, Reward: -1

Warehouse Robot (Stochastic: False)



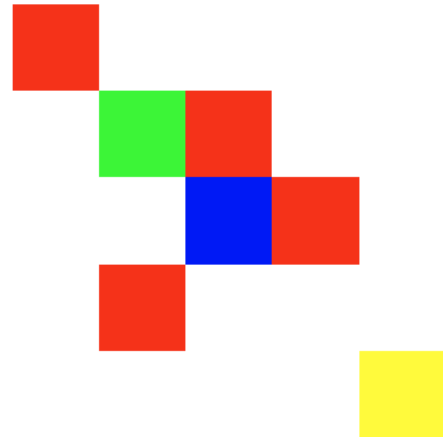
Action: Pickup, Reward: 25

Warehouse Robot (Stochastic: False)



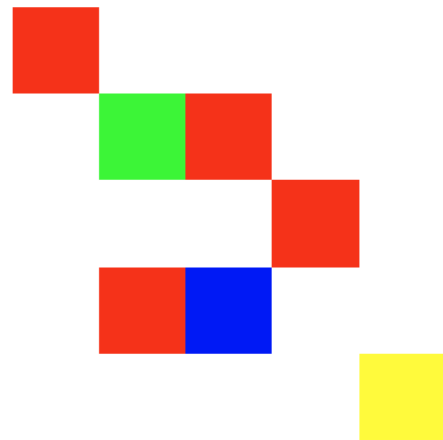
Action: Down, Reward: -1

Warehouse Robot (Stochastic: False)



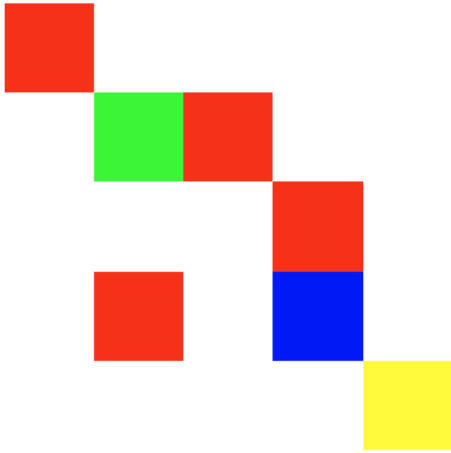
Action: Right, Reward: -1

Warehouse Robot (Stochastic: False)



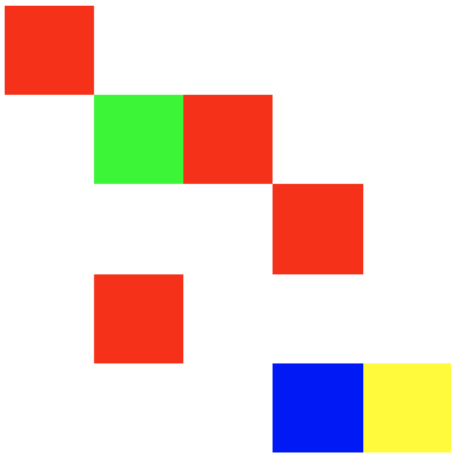
Action: Down, Reward: -1

Warehouse Robot (Stochastic: False)



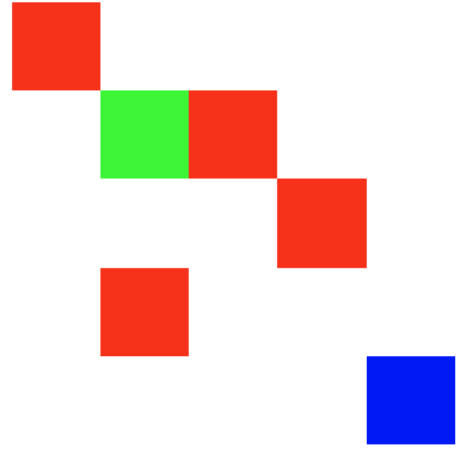
Action: Right, Reward: -1

Warehouse Robot (Stochastic: False)



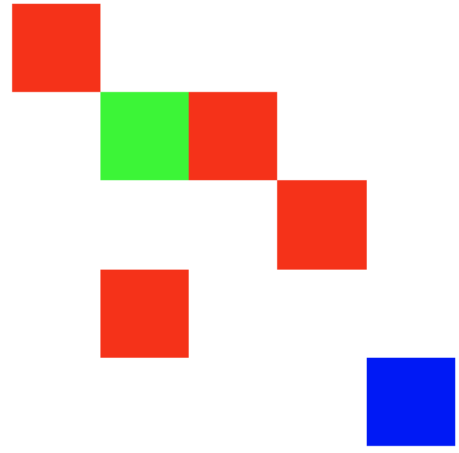
Action: Down, Reward: -1

Warehouse Robot (Stochastic: False)



Action: Right, Reward: -1

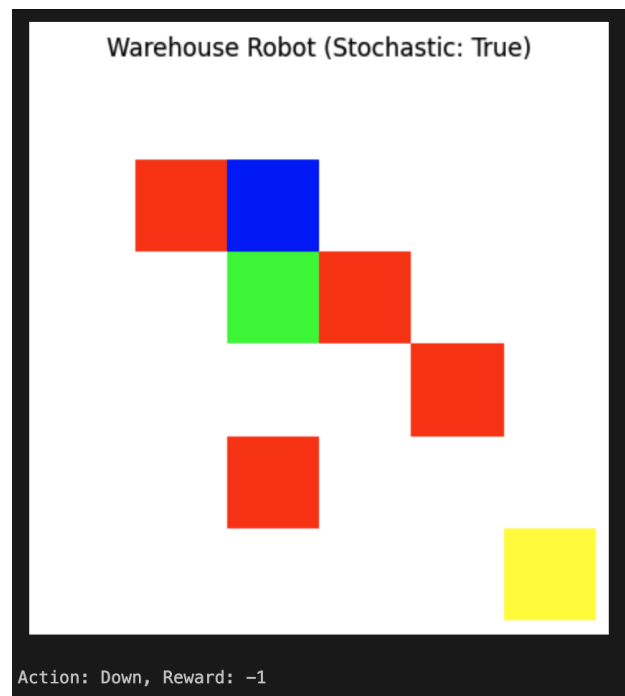
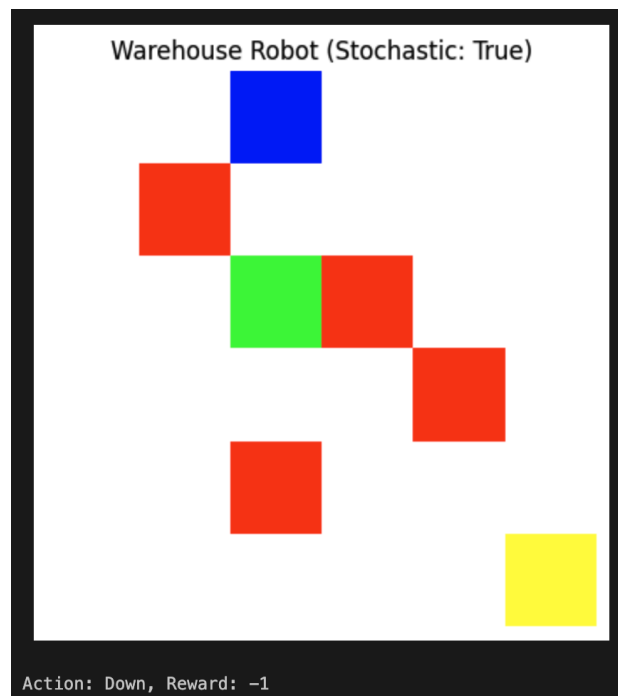
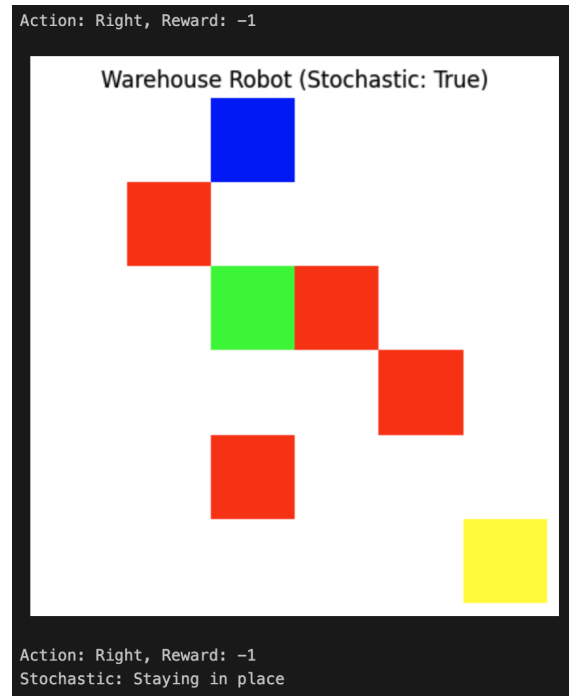
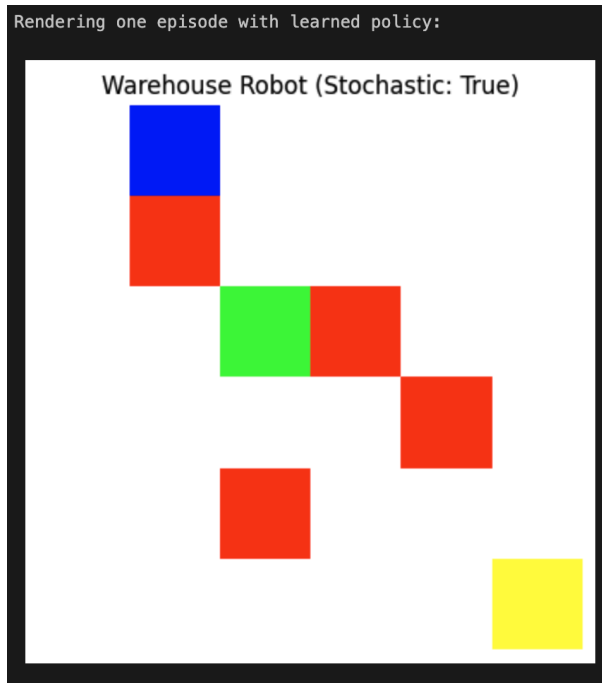
Warehouse Robot (Stochastic: False)



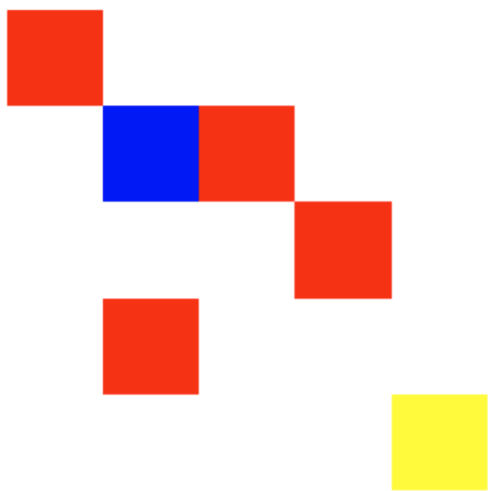
Action: Dropoff, Reward: 100
Task Completed!

STOCHASTIC ENVIRONMENT:

The screenshot below demonstrates the agent using greedy actions derived from its learned policy to reach the target position.

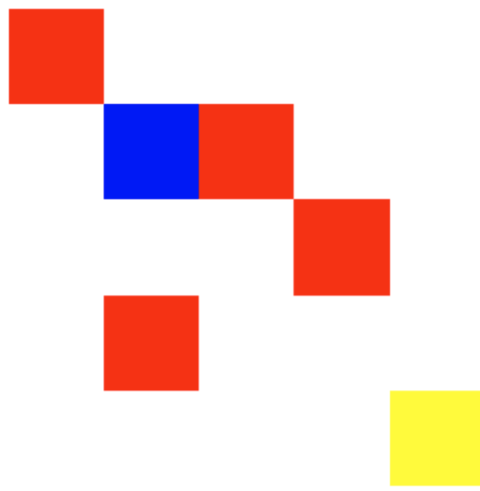


Warehouse Robot (Stochastic: True)



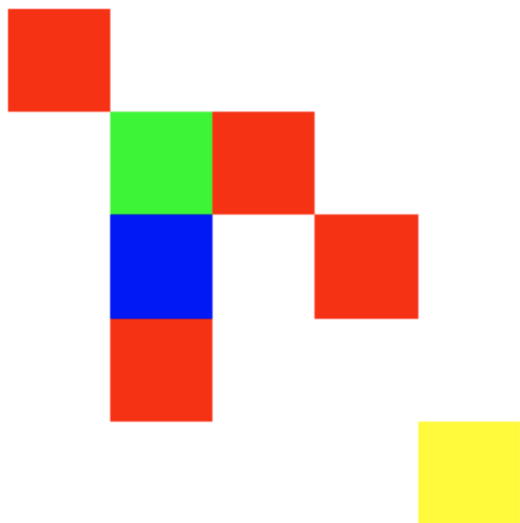
Action: Down, Reward: -1

Warehouse Robot (Stochastic: True)



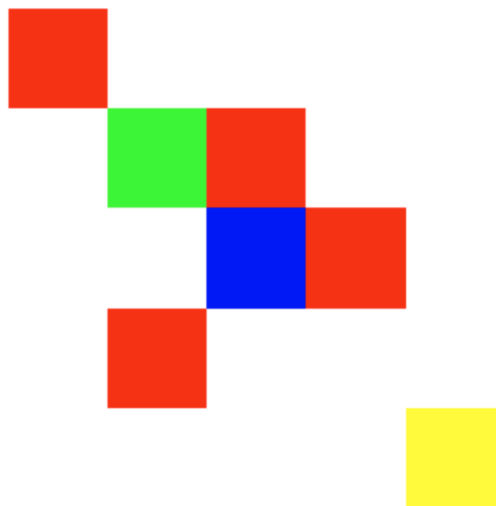
Action: Pickup, Reward: 25

Warehouse Robot (Stochastic: True)

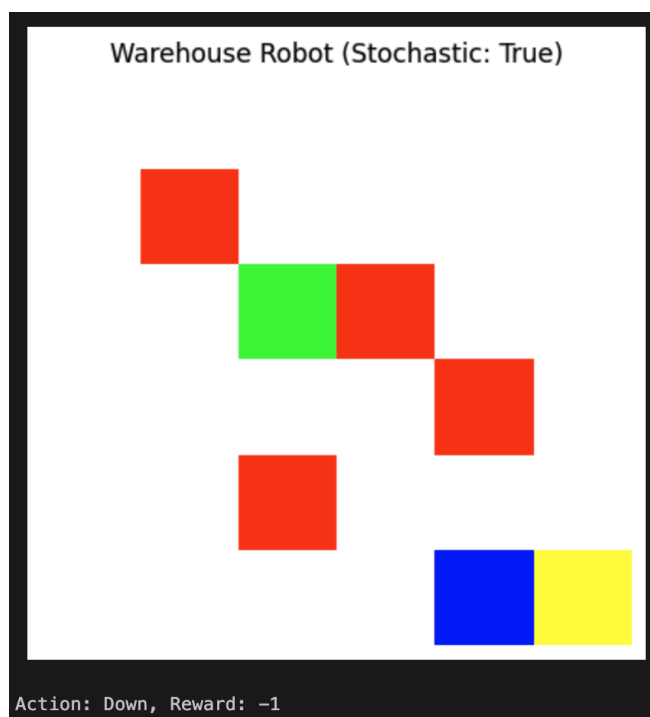
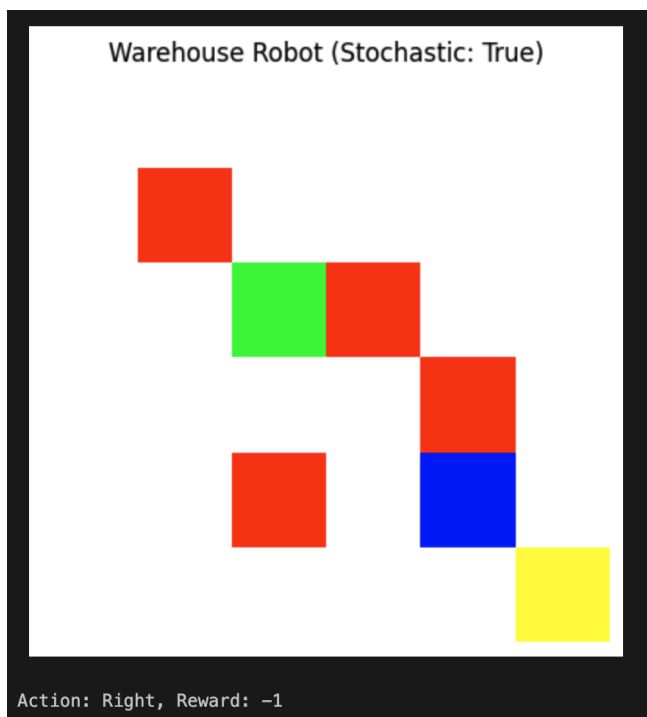
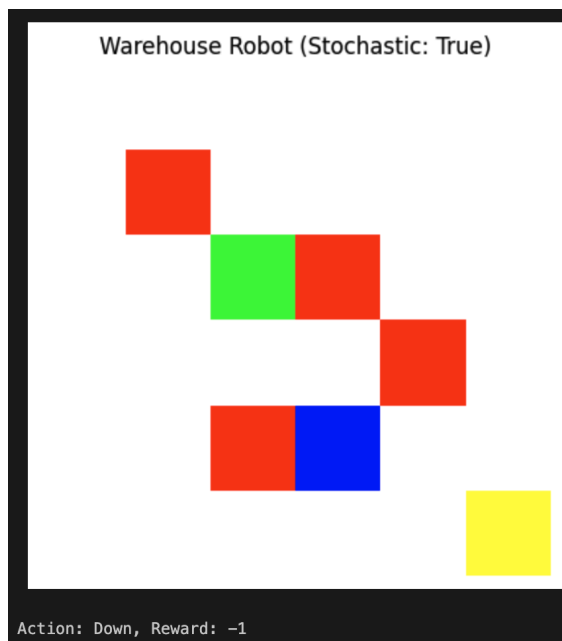
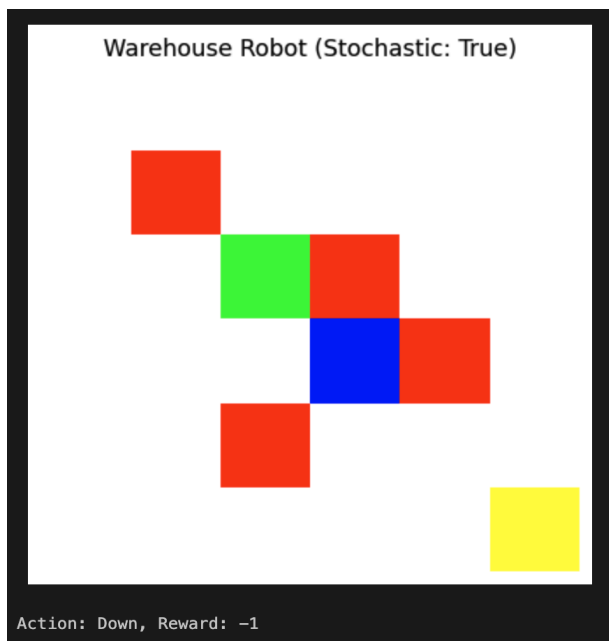


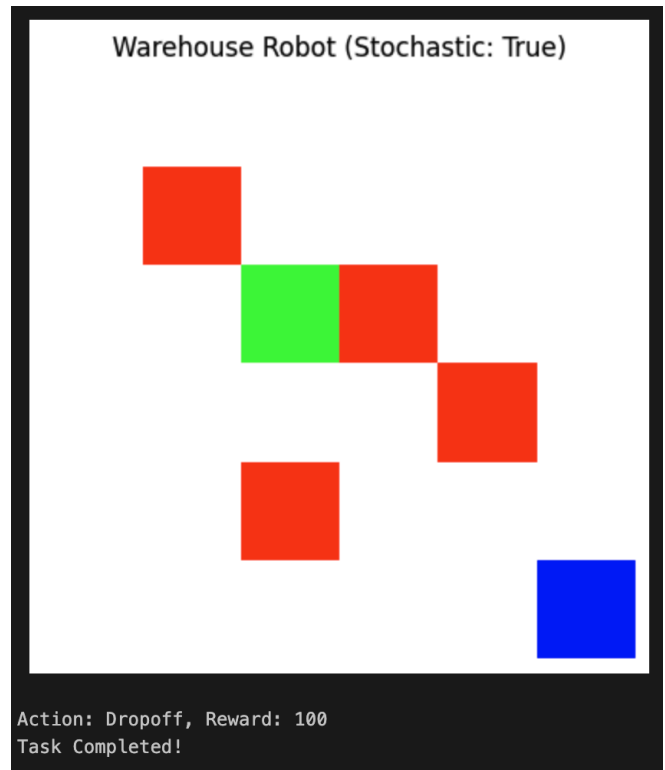
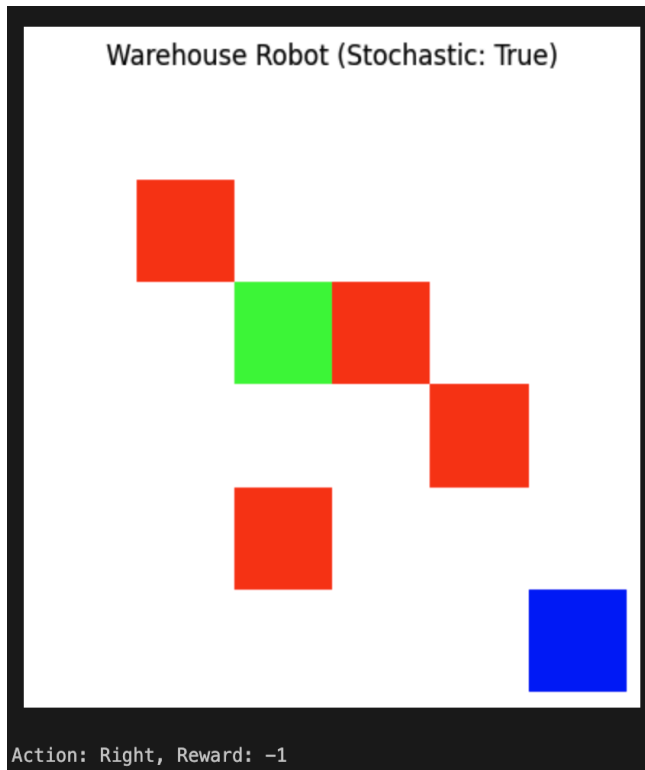
Action: Down, Reward: -1

Warehouse Robot (Stochastic: True)



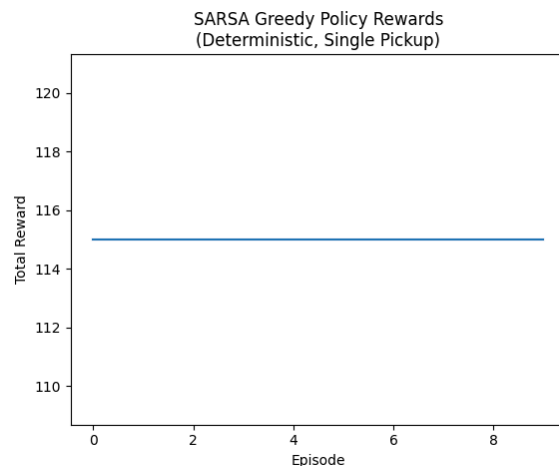
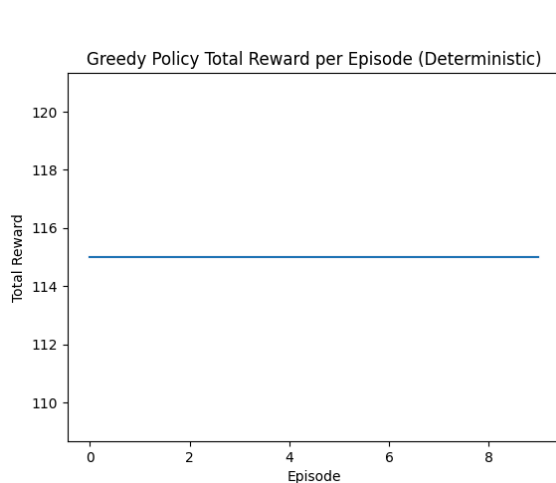
Action: Right, Reward: -1
Stochastic: Staying in place





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.

The comparison of the Greedy Policy and SARSA Policy in the deterministic environment is visualized in the below graph, where both policies achieve identical total rewards of 116 across all episodes, while the average reward is 112.67 for SARSA and slightly greater in Q learning 115.



The graphs compare the performance of Q-learning and SARSA in a deterministic environment, where both algorithms achieve consistent total rewards across episodes.

The key observations and interpretations are as follows:

1. Performance Consistency:

- Both Q-learning and SARSA yield a constant total reward of 115 across all episodes in the deterministic environment.
- This indicates that both algorithms successfully converge to an optimal or near-optimal policy, achieving identical performance under deterministic conditions.

2. Reward Dynamics:

- The flat lines in the graphs demonstrate that there is no variation in rewards over episodes for either algorithm. This is expected in a deterministic environment where the outcomes of actions are predictable, and both algorithms can reliably learn the optimal policy.

3. Algorithm Behavior:

- Q-learning, being an *off-policy* algorithm, learns the optimal policy by exploring actions independently of the current policy.
- SARSA, an *on-policy* algorithm, learns the policy based on the actions it actually takes. Despite this difference, their performance converges to the same level in this deterministic setup because exploration dynamics are less critical when outcomes are fixed.

SARSA Algorithm:

```
Action: Dropoff, Reward: 100
Task Completed!
SARSA Episode 0, Reward: -404, Epsilon: 0.995
SARSA Episode 100, Reward: -233, Epsilon: 0.6027415843082742
SARSA Episode 200, Reward: 84, Epsilon: 0.36512303261753626
SARSA Episode 300, Reward: 111, Epsilon: 0.2211807388415433
SARSA Episode 400, Reward: 115, Epsilon: 0.13398475271138335
SARSA Episode 0, Reward: -309, Epsilon: 0.995
SARSA Episode 100, Reward: -151, Epsilon: 0.6027415843082742
SARSA Episode 200, Reward: 77, Epsilon: 0.36512303261753626
SARSA Episode 300, Reward: 108, Epsilon: 0.2211807388415433
SARSA Episode 400, Reward: 108, Epsilon: 0.13398475271138335
SARSA Episode 0, Reward: -290, Epsilon: 0.995
SARSA Episode 100, Reward: -166, Epsilon: 0.6027415843082742
SARSA Episode 200, Reward: -56, Epsilon: 0.36512303261753626
SARSA Episode 300, Reward: 100, Epsilon: 0.2211807388415433
SARSA Episode 400, Reward: 91, Epsilon: 0.13398475271138335
SARSA Episode 0, Reward: -157, Epsilon: 0.995
SARSA Episode 100, Reward: -203, Epsilon: 0.6027415843082742
SARSA Episode 200, Reward: 107, Epsilon: 0.36512303261753626
SARSA Episode 300, Reward: 114, Epsilon: 0.2211807388415433
SARSA Episode 400, Reward: 112, Epsilon: 0.13398475271138335
SARSA Episode 0, Reward: -271, Epsilon: 0.995
SARSA Episode 100, Reward: -200, Epsilon: 0.6027415843082742
SARSA Episode 200, Reward: -80, Epsilon: 0.36512303261753626
...
SARSA Episode 400, Reward: 99, Epsilon: 0.06699237635569168
alpha: 0.9, gamma: 0.99, epsilon: 0.5, score: -12.002999999999997

Best hyperparameters: {'alpha': 0.1, 'gamma': 0.95, 'epsilon': 0.5} with average reward: 112.672
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

Q learning algorithm:

```
Best hyperparameters found:
Average Reward: 115.0
gamma: 0.9687366821547194
epsilon_decay: 0.9542302053745603
alpha: 0.25061550071976224
Episode 0, Reward: -295, Epsilon: 0.9542302053745603
Episode 100, Reward: 115, Epsilon: 0.01
Episode 200, Reward: 115, Epsilon: 0.01
Episode 300, Reward: 115, Epsilon: 0.01
Episode 400, Reward: 114, Epsilon: 0.01
Episode 500, Reward: 115, Epsilon: 0.01
Episode 600, Reward: 113, Epsilon: 0.01
Episode 700, Reward: 115, Epsilon: 0.01
Episode 800, Reward: 115, Epsilon: 0.01
Episode 900, Reward: 115, Epsilon: 0.01
```

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.

The performance of SARSA and Q-learning in the same stochastic environment can be compared using graphs showing their respective reward dynamics across episodes.

1. *Reward Dynamics:*

- Both SARSA and Q-learning show fluctuations in total rewards across episodes, reflecting the stochastic nature of the environment.
- The fluctuations are more visible in Q-learning compared to SARSA, showing that Q-learning's off-policy nature makes it more sensitive to variations in the environment.

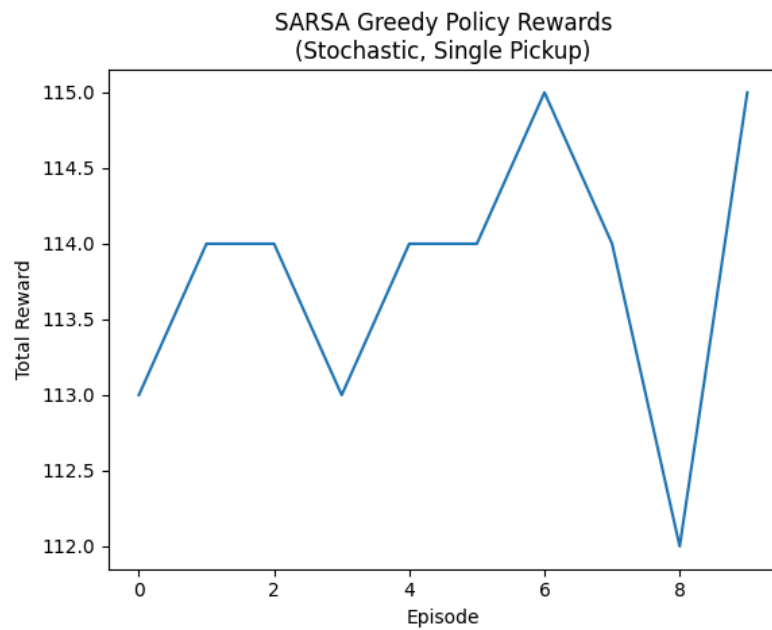
2. *Average Reward:*

- Both algorithms achieve similar average rewards over time, suggesting that they converge to comparable policies despite their different learning approaches.
- However, SARSA appears slightly more stable, with fewer extreme drops in total rewards.

3. *Exploration vs. Exploitation:*

- SARSA's on-policy strategy (learning based on the agent's actual actions) results in a more cautious approach, leading to smoother transitions and fewer drastic changes in rewards.

- Q-learning's off-policy strategy (learning based on the maximum possible future reward) is more aggressive, often leading to higher variance in rewards as it explores riskier actions.



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

Tabular methods in reinforcement learning (RL) are techniques that use a table (or array) to store value estimates for each state or state-action pair, relying on exact representations rather than approximations. These methods are well-suited for problems with discrete, finite state and action spaces, like gridworlds or simple games. Below tabular methods, focusing on Q-Learning and SARSA and Value Iteration, with their update functions and features.

In the given problem assignment, I have implemented Q learning and SARSA algorithm. Both are tabular methods.

SARSA(State-Action-Reward-State-Action) -

Update Function:

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

Key Features:

- On-policy: Updates reflect the policy's behavior (e.g., exploration included), making it more conservative.
- Converges to the optimal policy if the policy becomes greedy in the limit with infinite exploration.
- Also model-free, like Q-Learning.

Q Learning Algorithm:

Update Function:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha (r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t))$$

Key Features:

- Off-policy: Updates based on the best possible future action, not the action actually taken next.
- Converges to the optimal policy given enough exploration (e.g., ϵ -greedy) and visits to all state-action pairs.
- No need for a model of the environment (rewards and transitions).

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

Goal : The rewards should align with the desired behavior, encouraging the robot to complete pickups and dropoffs efficiently.

Penalty for Undesired Actions: Negative rewards for wrong steps or collisions discourage inefficient or dangerous behavior.

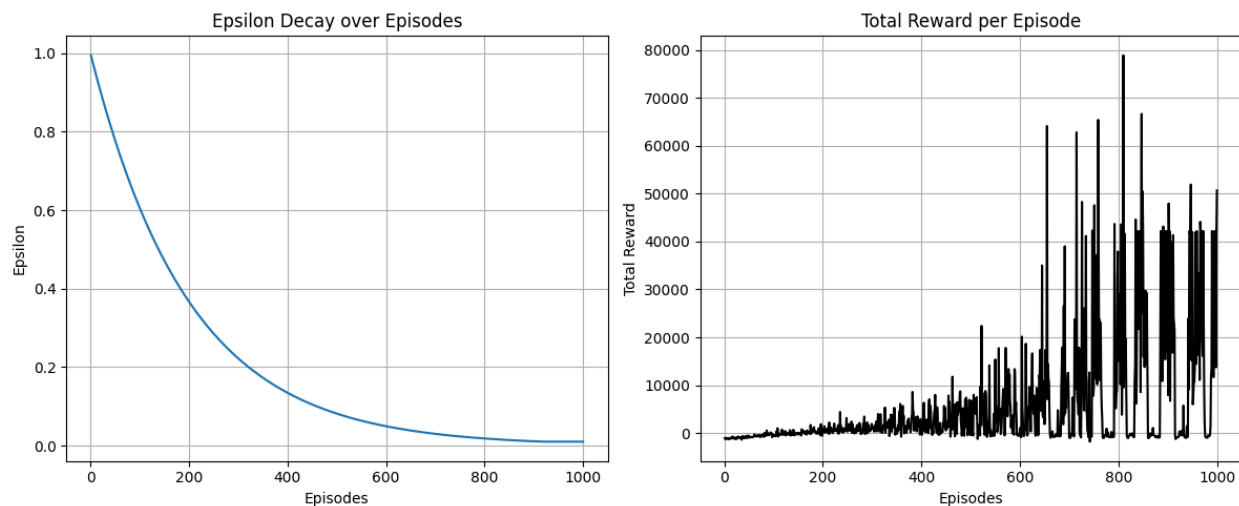
Scalability: The reward structure should work well for both single and multiple pickup scenarios.

Proportionality: Rewards should be proportional to the importance of each action.

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.

The provided plots show the results of applying the Q-learning algorithm for 1000 episodes. The two graphs represent Epsilon Decay over Episodes and Total Reward per Episode, which are key indicators of the learning process in Q-learning.



Results:

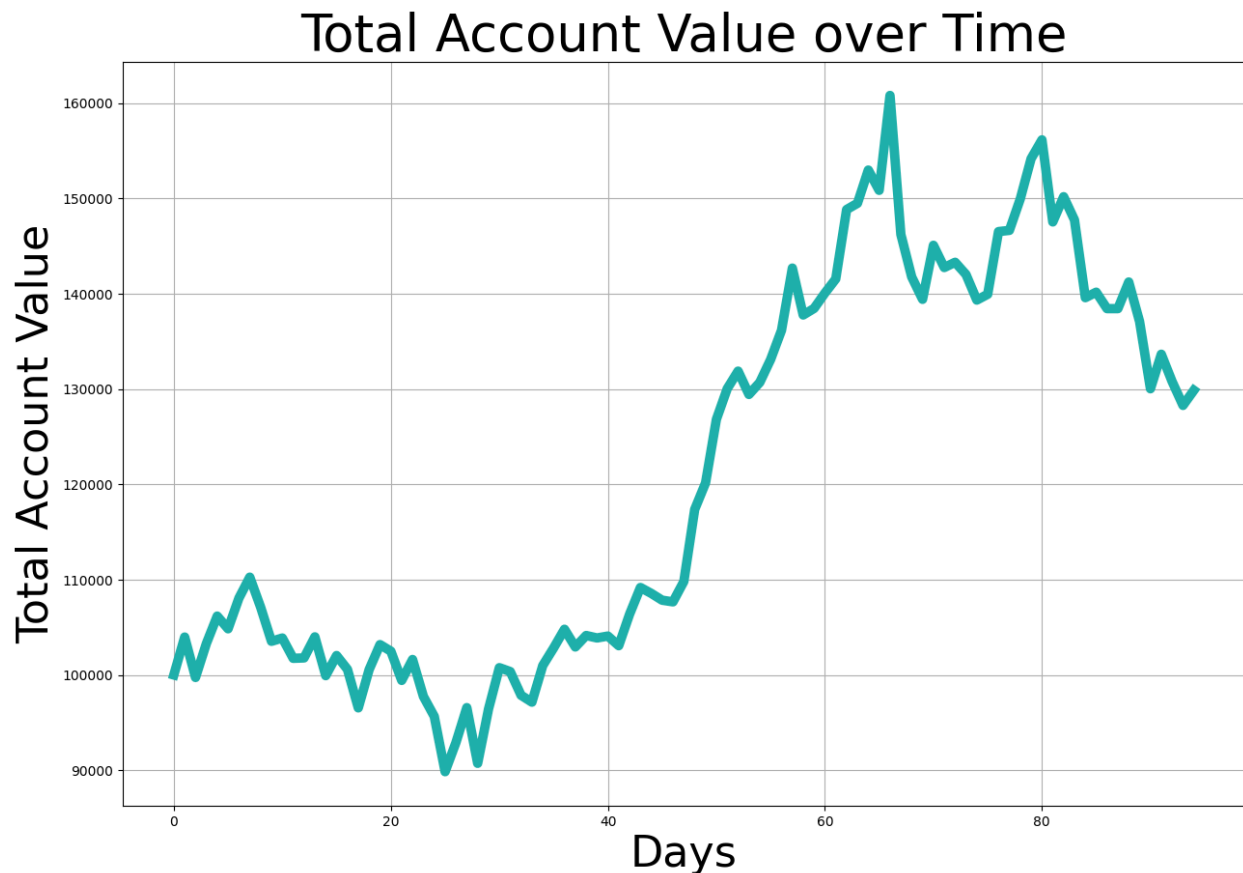
- **Epsilon Decay:** The smooth decay of epsilon ensures a balanced transition between exploration and exploitation. This is crucial for discovering optimal strategies while avoiding getting stuck in local optima.
- **Reward Growth:** The increasing trend in total rewards demonstrates that the Q-learning algorithm is successfully learning a better policy over time. However, fluctuations

suggest that further tuning (e.g., adjusting learning rate or reward function) might help stabilize performance.

- In a stock trading context, these results indicate that the agent is progressively learning to make profitable trades but still encounters challenges due to market complexity or sub-optimal parameter settings.

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.

The evaluation of the trained agent's performance, using only greedy actions from the learned policy and with the training parameter set to 'False', is presented below:



1. Training Performance:

- The training log shows the agent's performance across 1000 episodes.
- The epsilon value (exploration rate) decreases steadily from 0.6058 in episode 100 to 0.0100 in episode 1000, shows a shift from exploration to exploitation.

- The total reward increases significantly over time, starting at 6.83 in episode 100 and reaching a high of 62393.23 in episode 1000, demonstrating that the agent learns an effective policy over time.
2. Evaluation Performance:
- During evaluation (where the train parameter is set to False), the agent selects actions greedily based on its learned policy.
 - The plot titled "Total Account Value over Time" illustrates the agent's account value progression during evaluation.
 - The account value starts at approximately 100,000 and exhibits a general upward trend with fluctuations. It peaks at around 160,000 before declining slightly toward the end of the evaluation period, stabilizing above 130,000.

Key Observations:

- The agent successfully learns a policy that increases the total account value over time during training and evaluation.
- Despite some volatility in performance (as seen in the plot), the overall trend indicates profitability.
- The decline after reaching a peak suggests potential overfitting to specific scenarios or market conditions, which could be addressed with further fine-tuning or additional training data.

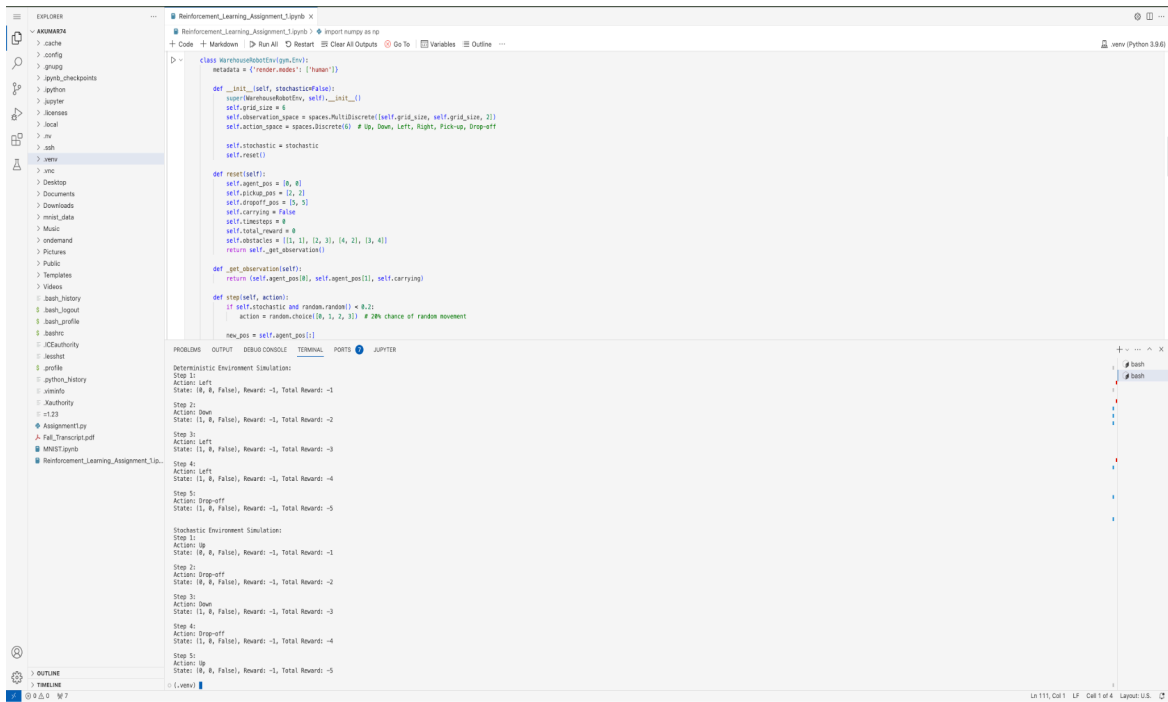
```
Training the Q-learning agent...
```

```
Episode 100/1000 completed. Epsilon: 0.6058, Total Reward: 6.83  
Episode 200/1000 completed. Epsilon: 0.3670, Total Reward: -150.25  
Episode 300/1000 completed. Epsilon: 0.2223, Total Reward: 1714.85  
Episode 400/1000 completed. Epsilon: 0.1347, Total Reward: 4077.68  
Episode 500/1000 completed. Epsilon: 0.0816, Total Reward: 3469.02  
Episode 600/1000 completed. Epsilon: 0.0494, Total Reward: 7603.46  
Episode 700/1000 completed. Epsilon: 0.0299, Total Reward: -253.97  
Episode 800/1000 completed. Epsilon: 0.0181, Total Reward: -778.88  
Episode 900/1000 completed. Epsilon: 0.0110, Total Reward: -471.57  
Episode 1000/1000 completed. Epsilon: 0.0100, Total Reward: 62393.23
```

Bonus Task:

CCR Submission:

Running the code in VS Code: The screenshot of the output below-



Git Submission:

