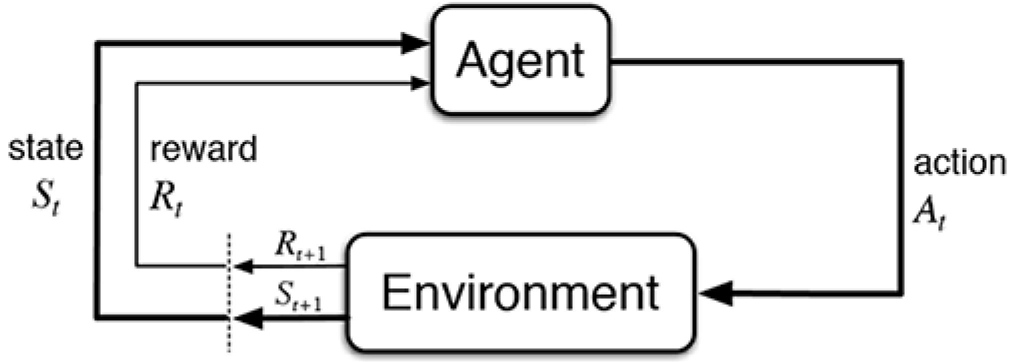
Classic Gym's Taxi Problem: A Reinforcement Learning Case Study

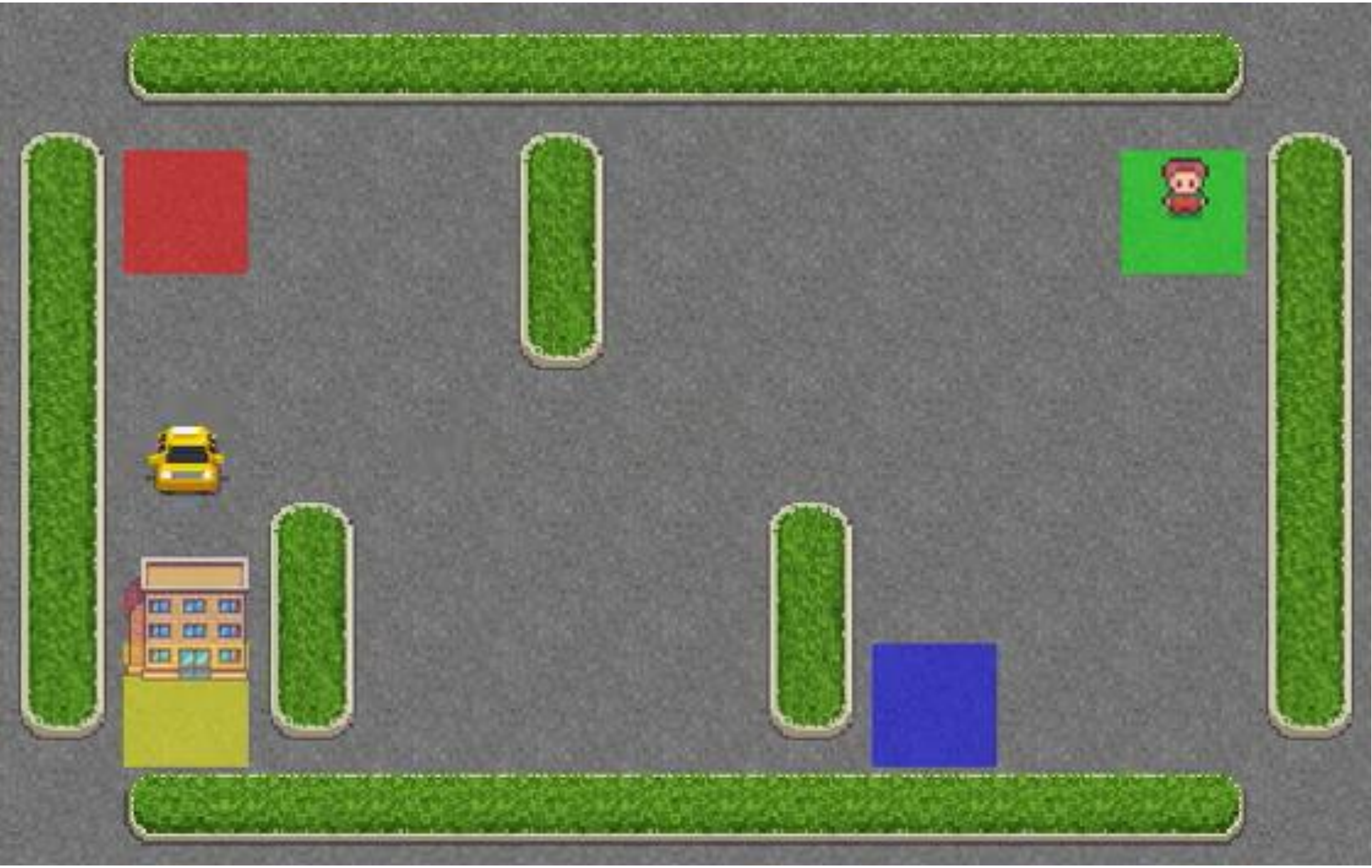
Solving the Taxi Problem using OpenAI Gym and Q-Learning

Introduction to Reinforcement Learning

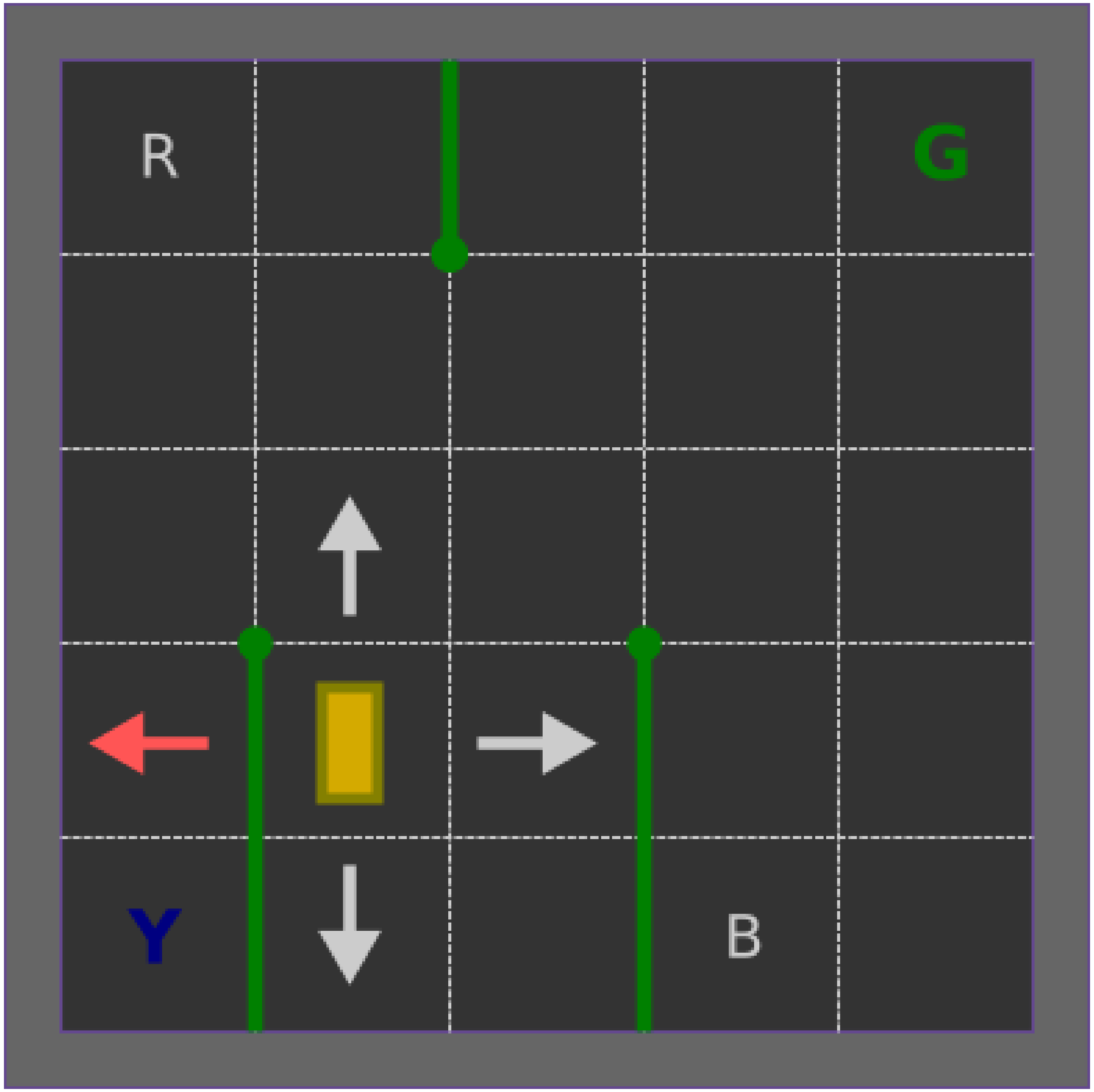
What is Reinforcement Learning (RL)?

- A type of machine learning where an agent learns to make decisions by interacting with an environment.
- The agent receives rewards or penalties based on its actions.
- Key concepts:
  - Agent, Environment, States, Actions, Rewards.





Need to design an algorithm to teach a taxi agent to navigate a small gridworld!



The Taxi Problem Overview

- What is the Taxi Problem?
  - An agent (taxi) needs to navigate a 5x5 grid to pick up and drop off passengers.
  - The taxi must pick up passengers from one location and drop them off at another.
  - The task is to maximize rewards by completing the job efficiently.

- State Elements:
  - Taxi's Location: Position in the 5x5 grid.
  - Passenger's Location: Pick-up and drop-off points.
  - Passenger’s State: Whether the passenger is in the taxi or not.

Action Space

- Possible Actions:
  - Move: Move North, South, East, or West.
  - Pick Up: Pick up the passenger from their location.
  - Drop Off: Drop off the passenger at the destination.

- Goal:
  - Maximize the total reward by performing actions that lead to successful passenger pick-up and drop-off.

Reward System

Rewards:

- +20 points for dropping off a passenger at the correct location.
- -1 point for each step taken (encouraging the agent to take fewer steps).
- -10 points for illegal actions (e.g., picking up a passenger when already carrying one).

Using OpenAI Gym

What is OpenAI Gym?

- A toolkit for developing and comparing RL algorithms.
- Provides various environments, including the Taxi-v3 environment.

Gym’s Taxi-v3 Environment:

- Predefined grid world with the agent, passenger, pick-up, and drop-off locations.
- Interaction via actions and receiving rewards.

Q-Learning for Solving the Taxi Problem

- Q-Learning Algorithm:
  - A model-free reinforcement learning algorithm.
  - Updates the Q-table based on state-action values to maximize rewards.
- Q-table:
  - A table with rows for states and columns for actions.
  - The Q-value is updated based on the formula:

$$\overset{\text{Q-value (for a state (S) and action(A))}}{\boxed{Q(S_t, A_t)}} \leftarrow \boxed{Q(S_t, A_t)} + \underset{\text{Learning rate}}{\boxed{\alpha}} \left[\overset{\text{Reward}}{\boxed{R_{t+1}}} + \underset{\text{Discount factor}}{\boxed{\gamma}} \overset{\text{Maximum expected future reward}}{\boxed{\max_a Q(S_{t+1}, a)}} - \boxed{Q(S_t, A_t)} \right]$$

Exploration vs. Exploitation

- Exploration:
  - Trying random actions to explore the environment.
- Exploitation:
  - Choosing the best-known action based on the Q-table.
- Epsilon-greedy:
  - A strategy to balance exploration and exploitation.

Training the Agent

- Training Process:
  - The agent is trained over multiple episodes.
  - It learns to navigate the grid, pick up passengers, and drop them off.
  - Updates the Q-table after each action taken.

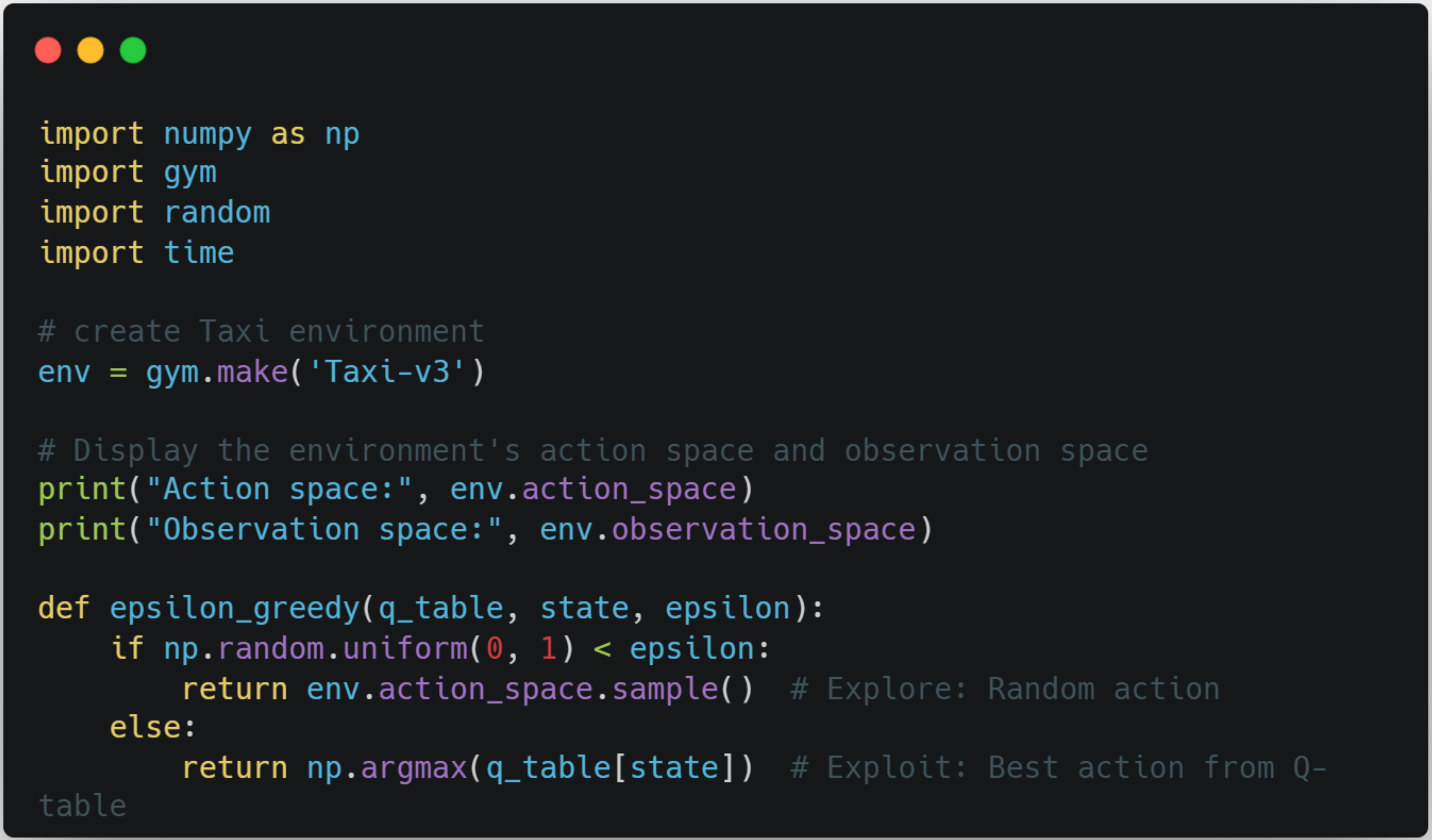
- Hyperparameters:
  - Alpha (Learning Rate): How much the Q-values are adjusted.
  - Gamma (Discount Factor): Future rewards’ importance.
  - Epsilon (Exploration Factor): Probability of exploration.

Testing the Trained Agent

- Post-Training:
  - After the agent is trained, test it by letting it follow the learned policy.
- Result:
  - The agent should be able to pick up passengers and drop them off efficiently with maximum rewards.

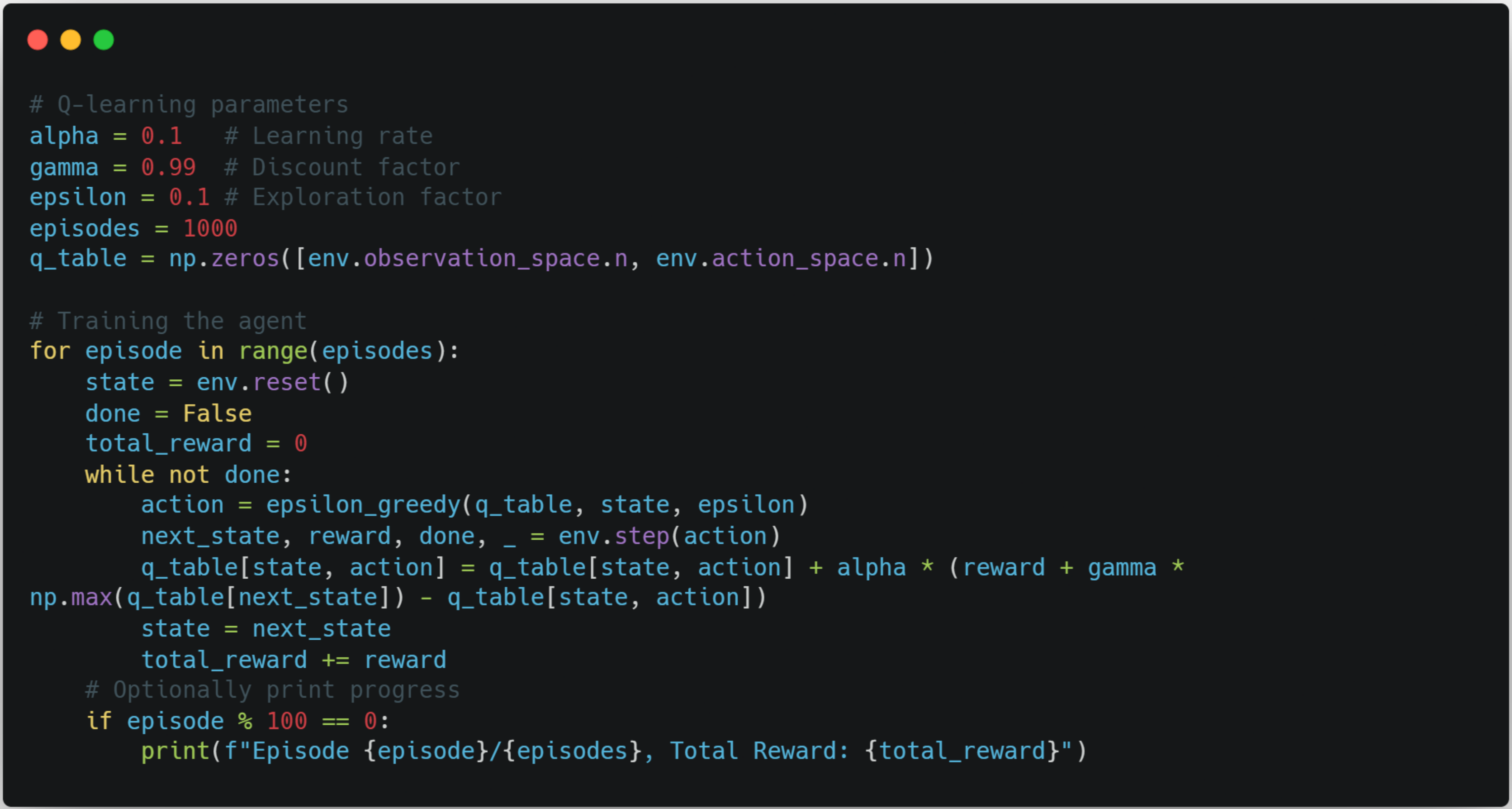
Hands-on!

Version: gym==0.16



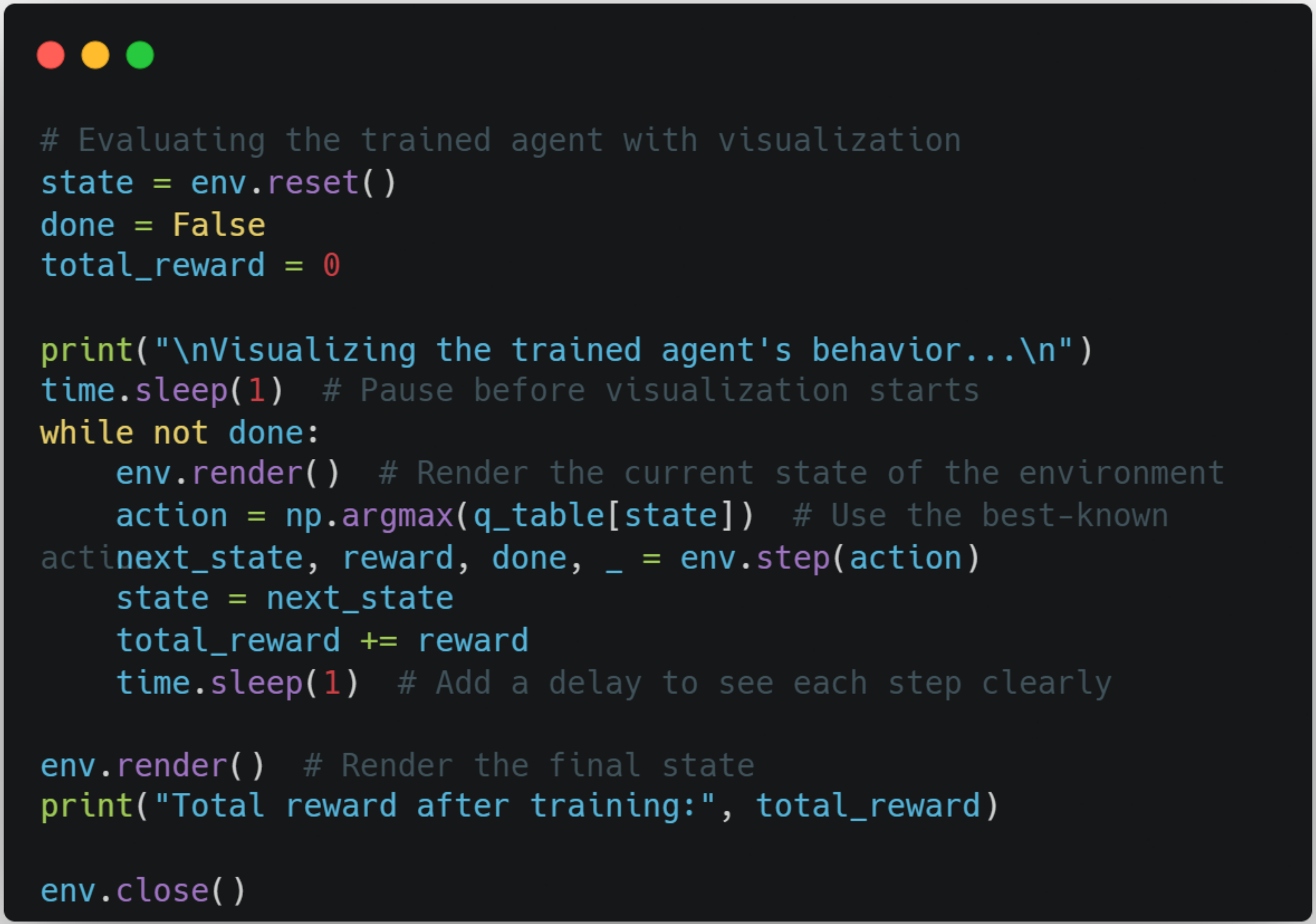
```python
import numpy as np
import gym
import random
import time

# create Taxi environment
env = gym.make('Taxi-v3')

# Display the environment's action space and observation space
print("Action space:", env.action_space)
print("Observation space:", env.observation_space)

def epsilon_greedy(q_table, state, epsilon):
    if np.random.uniform(0, 1) < epsilon:
        return env.action_space.sample()  # Explore: Random action
    else:
        return np.argmax(q_table[state])  # Exploit: Best action from Q-
table
```

Hands-on!



```python
# Q-learning parameters
alpha = 0.1   # Learning rate
gamma = 0.99  # Discount factor
epsilon = 0.1 # Exploration factor
episodes = 1000
q_table = np.zeros([env.observation_space.n, env.action_space.n])

# Training the agent
for episode in range(episodes):
    state = env.reset()
    done = False
    total_reward = 0
    while not done:
        action = epsilon_greedy(q_table, state, epsilon)
        next_state, reward, done, _ = env.step(action)
        q_table[state, action] = q_table[state, action] + alpha * (reward + gamma *
np.max(q_table[next_state]) - q_table[state, action])
        state = next_state
        total_reward += reward
    # Optionally print progress
    if episode % 100 == 0:
        print(f"Episode {episode}/{episodes}, Total Reward: {total_reward}")
```

Hands-on!

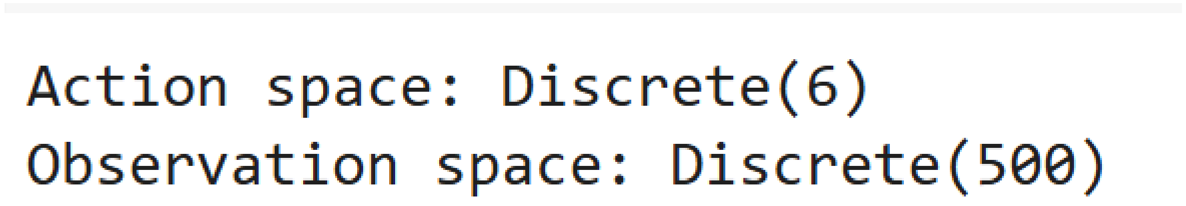


```
# Evaluating the trained agent with visualization
state = env.reset()
done = False
total_reward = 0

print("\nVisualizing the trained agent's behavior...\n")
time.sleep(1)  # Pause before visualization starts
while not done:
    env.render()  # Render the current state of the environment
    action = np.argmax(q_table[state])  # Use the best-known
actinext_state, reward, done, _ = env.step(action)
    state = next_state
    total_reward += reward
    time.sleep(1)  # Add a delay to see each step clearly

env.render()  # Render the final state
print("Total reward after training:", total_reward)

env.close()
```

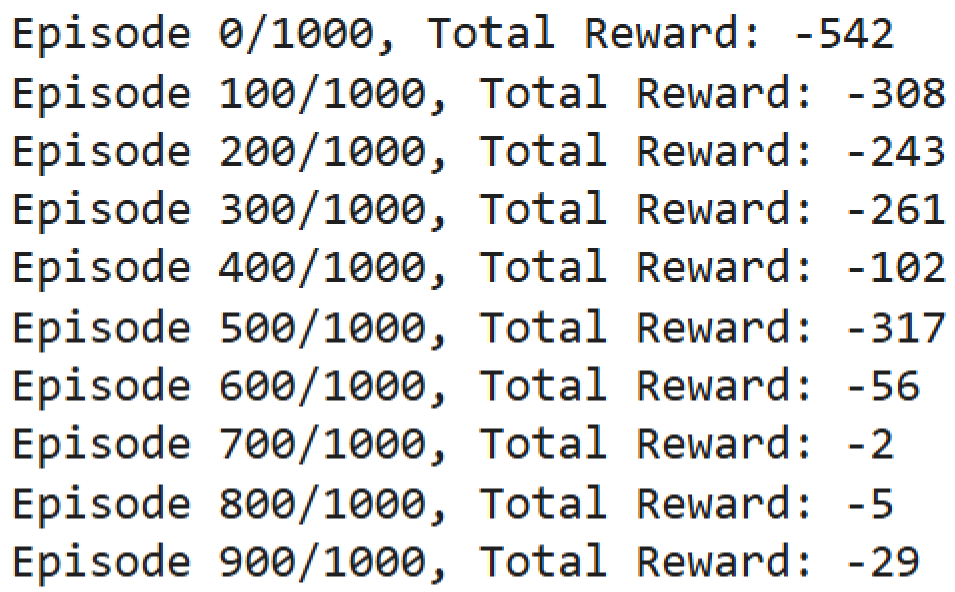
Results

- Training:
  - After several episodes, the agent learns the best path for picking up and dropping off passengers.
- Testing:
  - The agent performs the task in an optimized manner based on the learned Q-values.

Results (Training)



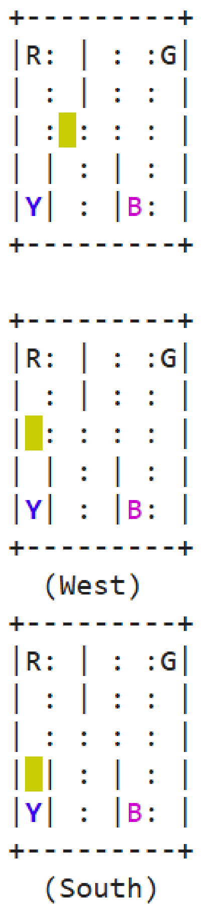
```
Action space: Discrete(6)
Observation space: Discrete(500)
```



```
Episode 0/1000, Total Reward: -542
Episode 100/1000, Total Reward: -308
Episode 200/1000, Total Reward: -243
Episode 300/1000, Total Reward: -261
Episode 400/1000, Total Reward: -102
Episode 500/1000, Total Reward: -317
Episode 600/1000, Total Reward: -56
Episode 700/1000, Total Reward: -2
Episode 800/1000, Total Reward: -5
Episode 900/1000, Total Reward: -29
```

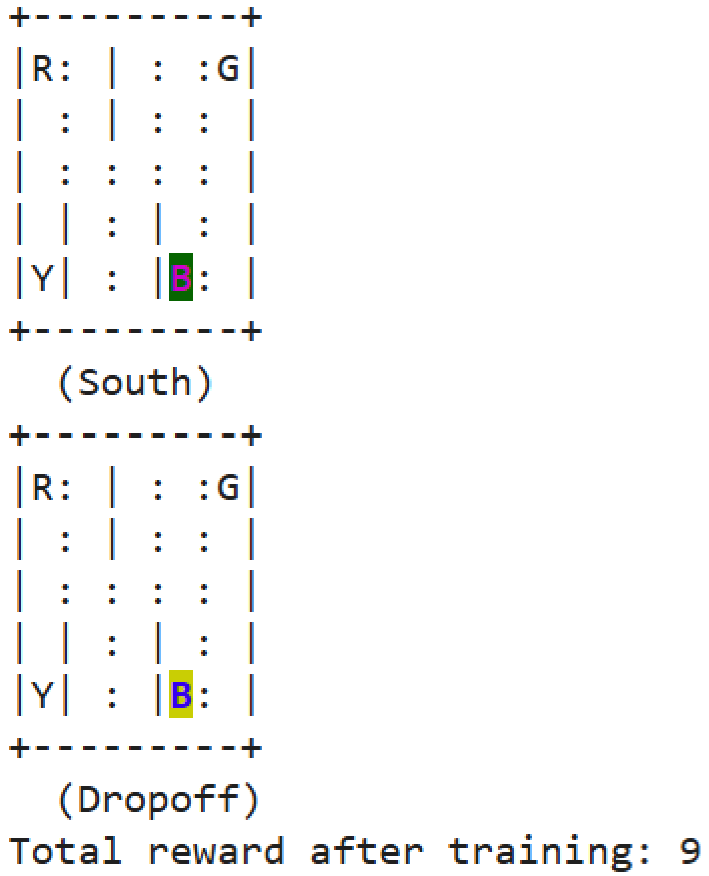
Results (Testing)

```
Visualizing the trained agent's behavior...
```



```
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+

+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (West)
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (South)
```

...



```
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (South)
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (Dropoff)
Total reward after training: 9
```

You Wanna Try It Too!?

💻 Check out this GitHub Repo:

Try this code and experiment!

https://github.com/MohanKrishnaGR/Reinforcement-Learning-Taxi-v3

GitHub

Conclusion

- Key Takeaways:
  - Reinforcement learning allows agents to learn optimal strategies through trial and error.
  - Q-learning is a fundamental RL algorithm that can be applied to simple environments like the Taxi problem.
  - OpenAI Gym provides an excellent platform to experiment with RL algorithms.

References

- Textbooks:
  - Sutton, R. S., & Barto, A. G. (2018). Reinforcement Learning: An Introduction (2nd ed.).
- Web References:
  - OpenAI Gym (https://gym.openai.com)
  - Q-Learning Algorithm (https://www.davidsilver.uk/)

THANK YOU!

To connect with me, you can find my professional profile and contact details on LinkedIn.

Feel free to reach out for collaborations, opportunities, or just to exchange ideas about the latest in software development and technology.

Let's innovate together!

LinkedIn Profile : https://www.linkedin.com/in/grmk/

GitHub Profile : https://github.com/MohanKrishnaGR