Faculty of Technology and Engineering

Chandubhai S. Patel Institute of Technology

Date: / /

Practical Performa

Academic Year	:	2025-26	Semester	:	7th
Course code	:	OCCSE4001	Course name	:	Reinforcement Learning

Practical- No. 3

Aim: To implement SARSA and Q-learning in a Grid World environment and compare their policies, showing that SARSA learns more conservative paths than Q-learning.

Code:

```
[1]: import gymnasium as gym
     import numpy as np
     import time
```

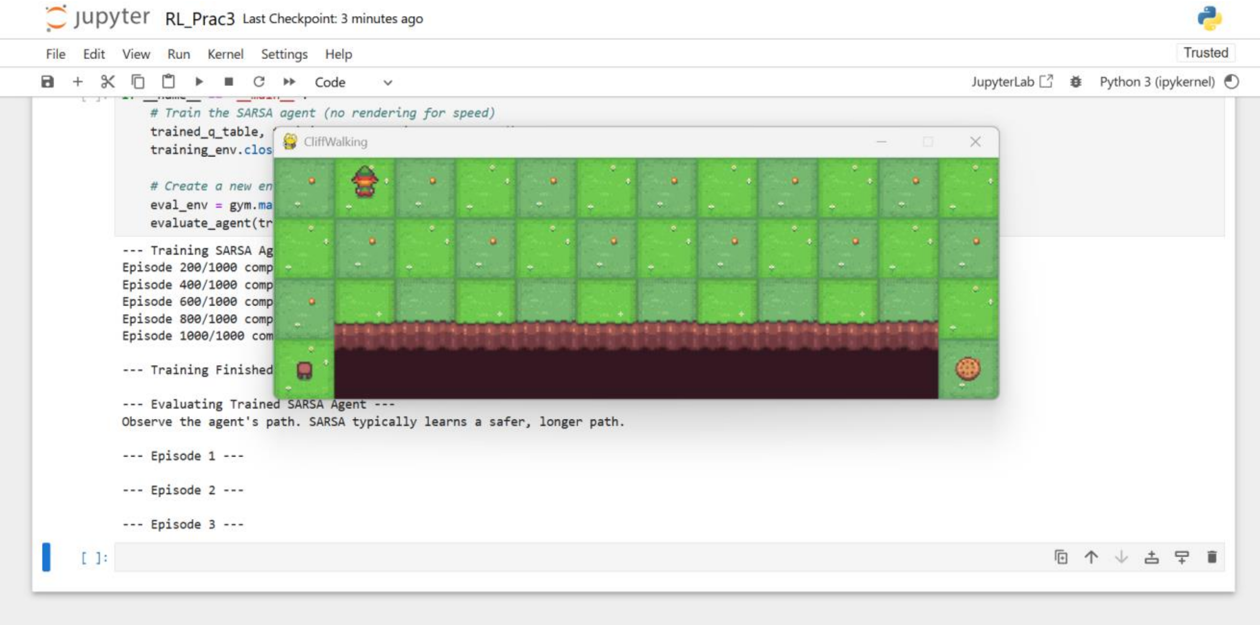
```
[2]: def train_sarsa_agent():
         # 1. Create the environment
         env = gym.make("CliffWalking-v1")
         state_space_size = env.observation_space.n
         action_space_size = env.action_space.n

         # 2. Initialize Q-table
         q_table = np.zeros((state_space_size, action_space_size))

         # 3. Hyperparameters
         total_episodes = 1000
         learning_rate = 0.5
         discount_factor = 0.95

         epsilon = 1.0
         max_epsilon = 1.0
         min_epsilon = 0.01
         epsilon_decay_rate = 0.005

         print("--- Training SARSA Agent ---")
```

```python
    # 4. Training loop
    for episode in range(total_episodes):
        state, _ = env.reset()
        done = False

        # Choose first action using epsilon-greedy
        if np.random.uniform(0, 1) < epsilon:
            action = env.action_space.sample()
        else:
            action = np.argmax(q_table[state, :])

        while not done:
            # Take action, get next state and reward
            next_state, reward, done, _, _ = env.step(action)

            # Choose the *next* action using epsilon-greedy for the next state
            if np.random.uniform(0, 1) < epsilon:
                next_action = env.action_space.sample()
            else:
                next_action = np.argmax(q_table[next_state, :])

            # SARSA update rule
            current_q = q_table[state, action]
            next_q = q_table[next_state, next_action] # The Q-value for the action we will actually take
            new_q = current_q + learning_rate * (reward + discount_factor * next_q - current_q)
            q_table[state, action] = new_q

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

        # Decay epsilon
        epsilon = min_epsilon + (max_epsilon - min_epsilon) * np.exp(-epsilon_decay_rate * episode)

        if (episode + 1) % 200 == 0:
            print(f"Episode {episode + 1}/{total_episodes} completed.")

    print("\n--- Training Finished ---")
    return q_table, env
```

[3]:
```python
def evaluate_agent(q_table, env, episodes=5):
    print("\n--- Evaluating Trained SARSA Agent ---")
    print("Observe the agent's path. SARSA typically learns a safer, longer path.")

    for episode in range(episodes):
        state, _ = env.reset()
        done = False
        print(f"\n--- Episode {episode+1} ---")
        time.sleep(1)

        while not done:
            # In evaluation, we always take the best action
            action = np.argmax(q_table[state, :])
            next_state, reward, done, _, _ = env.step(action)
            state = next_state

            # Render for visualization
            env.render()
            time.sleep(0.4)

    env.close()
```

[4]:
```python
if __name__ == "__main__":
    # Train the SARSA agent (no rendering for speed)
    trained_q_table, training_env = train_sarsa_agent()
    training_env.close()

    # Create a new environment with rendering to watch the trained agent
    eval_env = gym.make("CliffWalking-v1", render_mode="human")
    evaluate_agent(trained_q_table, eval_env)
```

Output:



Grade/Marks **Sign of Lab Teacher with Date**

(_____ / 10)