AI6101 Introduction to AI and AI Ethics

Reinforcement Learning Assignment

Nanyang Technological University
School of Computer Science and Engineering
Master of Science in Artificial Intelligence
NTAMBARA Etienne
Matriculation No: G2304253K

AY 2023-2024 Semester 1

August 7, 2024

1 Introduction

In this report, I prefer to implement one of reinforcement Learning Algorithm for the CliffBoxPush- ing grid world game, which is a Q-learning algorithm, which is one of the algorithms used in Reinforcement learning where the agent (A) and learns by interacting with the environment called CliffBoxPushing in our assignment to obtain the optimal strategy for achieving- in the goal. It learns from the environment and re- ward system to make better decisions for each state of the game.[3]

2 Objective

The primary objective is to train an agent (A) that can navigate the grid world (_), pushing a box (B) towards the goal (G) without falling into the cliff (X) or letting the box fall. After the agent pushes the box to a goal state agent will be awarded according to the whole steps from the state is ((5, 0), (4, 1)) where (5, 0) is the position of the agent and (4, 1) is the position of the box to the state (3,12) for the agent and (4,13) for box at this stage game will be over with some rewards for the optimal policy.

3 Problem Format

1. State: Position of the agent and the box. example, at time step 0, the state is ((5, 0), (4,1))
2. Action: [up, down, left, right] equivalent to [1, 2, 3, 4]
3. Rewards:
   (a) -1 at each timestep
   (b) Negative distance value between the box and the goal (distance from box to goal)
   (c) Negative distance value between the agent and the box (distance from agent to box)
   (d) -1000 for an agent or box fall into a cliff
   (e) +1000 for a box reaches in the goal position (4,13) with the help of an agent.

4 Q-Learning Algorithm Implementation

Let's now explain the every concept used to train my RLAgent model especial in **learn** method where the assignment were requesting to add the modifications to solve the CliffBoxPushing grid-world game. Given codes has important parameters such as:

env, num_episodes, epsilon=0.1, alpha=0.1, gamma=0.99.

4.1 Meaning for each parameter

1. **env argument:** this refers to environment in the context of reinforcement learning, in this case our game envirinment is called CliffBox-Pushing where the agent select an optimal policy and push agent to the goal.

2. **num_episode:** this argument refers to a collection of states, actions, and rewards, which ends in a terminal state called goal (G) located at state (4,13)

3. **epsilon=0.1:** This parameter in reinforcement learning is used for epsilon-greedy policy.An epsilon-greedy strategy selects the best-known action with probability (1 - epsilon) and a random action with probability epsilon. in this case I used epsilon with value 0.1 which produced the best rewards in our training.

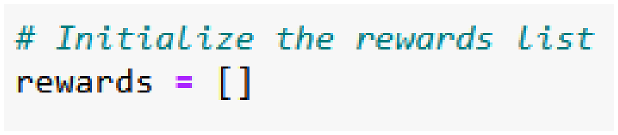
4. **alpha=0.1:** This is also called learning rate where it has a value of 0.1 in our task and this value produced the best reward in the training. In the context of Q-learning, for example, alpha decides how much new information will overrule previous information.[2]

5. **gamma=0.99:** Another option that has a default value of 0.99. Gamma, also known as the **discount factor**, is used in reinforcement learning algorithms to calculate the current value of future rewards.

4.2 Learn Method

Learn within a class called RLAgent, this method trains the agent based on interactions with an environment.

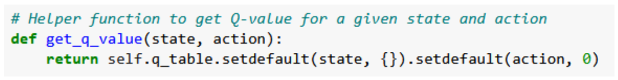
1. **Rewards:** it is Initialized Rewards in a list named rewards to hold the cumulative rewards earned by the agent in each episode.



```python
# Initialize the rewards list
rewards = []
```

Figure 1: Initialing Rewards

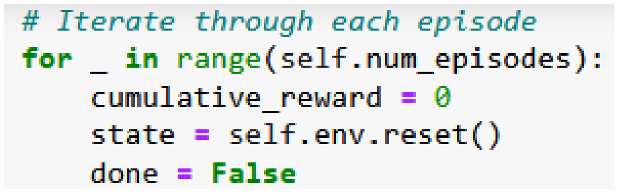
2. **get_q_value:** This function retrieves the Q-value for a particular state and action from the Q-table. If the state or action not found in the Q-table, it defaults to an empty dictionary for the state and a value of 0 for the action.



```python
# Helper function to get Q-value for a given state and action
def get_q_value(state, action):
    return self.q_table.setdefault(state, {}).setdefault(action, 0)
```

Figure 2: get q Value

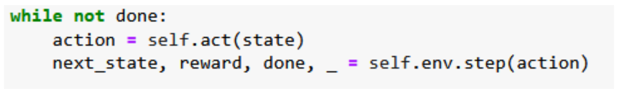
3. **Train a loop:** The agent trained across the number of episodes indicated by **self.num_episodes**. And also, **Environment Initialization**: At the start of each episode, the environment is reset to an initial state, and the done flag, which indicates whether or not the episode has been completed, is set to False.



```python
# Iterate through each episode
for _ in range(self.num_episodes):
    cumulative_reward = 0
    state = self.env.reset()
    done = False
```

Figure 3: Train Loop

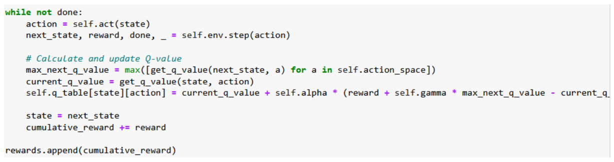
4. **Taking Actions in the Environment:** The agent uses the self.act(state) method to take an action depending on the current state. This action sent to the environment's step method, which provides the next state, reward, done flag, and maybe additional information (ignored with _).



```
while not done:
    action = self.act(state)
    next_state, reward, done, _ = self.env.step(action)
```

Figure 4: Taking Actions in the Environment

5. **Computing:** Q-Value computed by calculating the maximum Q-Value for the Next State: The maximum Q-value of the following state is required by the Q-learning process. This value is calculated by iterating over all possible actions.



```
while not done:
    action = self.act(state)
    next_state, reward, done, _ = self.env.step(action)

    # Calculate and update Q-value
    max_next_q_value = max([get_q_value(next_state, a) for a in self.action_space])
    current_q_value = get_q_value(state, action)
    self.q_table[state][action] = current_q_value + self.alpha * (reward + self.gamma * max_next_q_value - current_q_

    state = next_state
    cumulative_reward += reward

rewards.append(cumulative_reward)
```

Figure 5: Computing

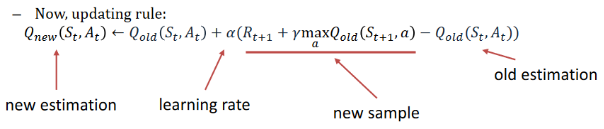
6. **Current Q-Value Fetching**: This fetches the current Q-value for the action taken in the current state.

7. **Updating the Q-Value**: The Q-learning algorithm's Q-value update rule is used here.

8. **State Transition and Reward Accumulation:** The agent enters the next state, and the reward for the activity performed is accumulated.

9. **Store Episode rewards**: When an episode concludes, the cumulative prize is added to the rewards list.

   Mathematical formula:



– Now, updating rule:

$$Q_{new}(S_t, A_t) \leftarrow Q_{old}(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_a Q_{old}(S_{t+1}, a) - Q_{old}(S_t, A_t))$$

Figure 6: Q-value computing.[1]

4.3 Learn Method all modified codes

```
def learn(self):

    # Initialize the rewards list
    rewards = []

    # Helper function to get Q-value for a given state and action
    def get_q_value(state, action):
        return self.q_table.setdefault(state, {}).setdefault(action, 0)

    # Print the best action and its associated Q-value for each state
    def print_policy():
        policy_with_rewards = {state: (max(actions, key=actions.get), max(actions.values())) for state, actions in self.q_table.items()}

        # Display the learned policy with expected rewards
        for state, (action, reward) in policy_with_rewards.items():
            print(f"State: {state}, Optimal action: {action}, Expected Rewards: {reward:.2f}")
        print("-" * 60)
```

```
            return policy_with_rewards

        # Iterate through each episode
        for _ in range(self.num_episodes):
            cumulative_reward = 0
            state = self.env.reset()
            done = False
            while not done:
                action = self.act(state)
                next_state, reward, done, _ = self.env.step(action)

                # Calculate and update Q-value
                max_next_q_value = max([get_q_value(next_state, a) for a in self.action_space])
                current_q_value = get_q_value(state, action)
                self.q_table[state][action] = current_q_value + self.alpha * (reward + self.gamma * max_next_q_value - current_q_value)

                state = next_state
                cumulative_reward += reward

            rewards.append(cumulative_reward)

        # Compute Average V-table
        print("AVERAGED V-TABLE: \n")
        shape = (self.env.world_height, self.env.world_width)
        avg_v_matrix = np.zeros(shape)
        count_matrix = np.zeros(shape, dtype=int)

        v_table = {state: max(actions.values()) for state, actions in self.q_table.items()}
        for state, value in v_table.items():
            agent_x, agent_y, _, _ = state
            avg_v_matrix[agent_x, agent_y] += value
            count_matrix[agent_x, agent_y] += 1

        avg_v_matrix = np.where(count_matrix != 0, avg_v_matrix / count_matrix, 0)

        for ax in range(shape[0]):
            row_values = [f"{avg_v_matrix[ax, ay]:.2f}" for ay in range(shape[1])]
            print(' | '.join(row_values))
        print("-" * 90)

        # Print V-table (the max Q-value for each state)
        print("V-TABLE FOR EACH STATE: \n")
        v_matrix = np.full((shape[0], shape[1], shape[0], shape[1]), -1000)
        for state, value in v_table.items():
            agent_x, agent_y, box_x, box_y = state
            v_matrix[agent_x, agent_y, box_x, box_y] = max(v_matrix[agent_x, agent_y, box_x, box_y], value)

        for ax in range(shape[0]):
            for ay in range(shape[1]):
                print(f"Agent at state ({ax}, {ay})")
                for bx in range(shape[0]):
                    row_values = [f"{v_matrix[ax, ay, bx, by]:.2f}" if v_matrix[ax, ay, bx, by] != -1000 else "0.00" for by in range(shape[1])]
                    print(' | '.join(row_values))
                print("-" * 90)

        print_policy()

        return rewards
```

4.4 Visualize Method all modified codes

```
def visualize(q_table):
    # Extract V-table from Q-table
    v_table = {state: max(actions.values()) for state, actions in q_table.items()}

    # Extract the policy (best action for each state)
    policy = {state: max(actions, key=actions.get) for state, actions in q_table.items()}

    # Display V-table
    print("V-TABLE:")
    for state, value in v_table.items():
        print(f"State: {state}, Value: {value:.2f}")

    # Display learned policy
    print("\nLEARNED POLICY:")
    for state, action in policy.items():
        print(f"State: {state}, Best Action: {action}")

# Test the function
agent = RLAgent(env, num_episodes=15000)
agent.learn()  # This updates the agent's Q-table
visualize(agent.q_table)
```

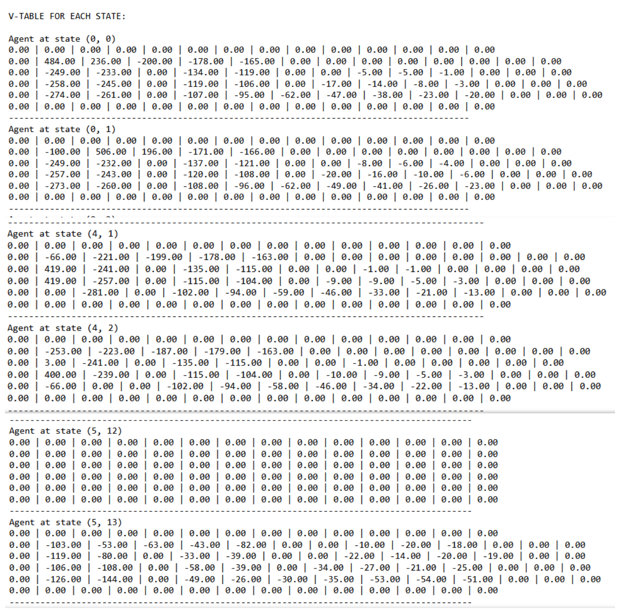
article graphicx

5 Output

5.1 Bug-free: correctly implement the code of your chosen RL algorithms and visualization

1. **V-TABLE AT EACH STATE**

   In reinforcement learning, the V-table (also known as the value table) displays the expected cumulative reward that an agent can acquire by starting from a specific state and then adopting the optimal policy. The value of each state in the V-table is a numerical representation of how "good" or "desirable" that state is for the agent in terms of maximizing its cumulative reward over the long term.



```
V-TABLE FOR EACH STATE:

Agent at state (0, 0)
0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00
0.00 | 484.00 | 236.00 | -200.00 | -178.00 | -165.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00
0.00 | -249.00 | -233.00 | 0.00 | -134.00 | -119.00 | 0.00 | 0.00 | -5.00 | -5.00 | -1.00 | 0.00 | 0.00 | 0.00
0.00 | -258.00 | -245.00 | 0.00 | -119.00 | -106.00 | 0.00 | -17.00 | -14.00 | -8.00 | -3.00 | 0.00 | 0.00 | 0.00
0.00 | -274.00 | -261.00 | 0.00 | -107.00 | -95.00 | -62.00 | -47.00 | -38.00 | -23.00 | -20.00 | 0.00 | 0.00 | 0.00
0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00
--------------------------------------------------------------------------------
Agent at state (0, 1)
0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00
0.00 | -100.00 | 506.00 | 196.00 | -171.00 | -166.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00
0.00 | -249.00 | -232.00 | 0.00 | -137.00 | -121.00 | 0.00 | 0.00 | -8.00 | -6.00 | -4.00 | 0.00 | 0.00 | 0.00
0.00 | -257.00 | -243.00 | 0.00 | -120.00 | -108.00 | 0.00 | -20.00 | -16.00 | -10.00 | -6.00 | 0.00 | 0.00 | 0.00
0.00 | -273.00 | -260.00 | 0.00 | -108.00 | -96.00 | -62.00 | -49.00 | -41.00 | -26.00 | -23.00 | 0.00 | 0.00 | 0.00
0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Agent at state (4, 1)
0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00
0.00 | -66.00 | -221.00 | -199.00 | -178.00 | -163.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00
0.00 | 419.00 | -241.00 | 0.00 | -135.00 | -115.00 | 0.00 | 0.00 | -1.00 | -1.00 | 0.00 | 0.00 | 0.00 | 0.00
0.00 | 419.00 | -257.00 | 0.00 | -115.00 | -104.00 | 0.00 | -9.00 | -9.00 | -5.00 | -3.00 | 0.00 | 0.00 | 0.00
0.00 | 0.00 | -281.00 | 0.00 | -102.00 | -94.00 | -59.00 | -46.00 | -33.00 | -21.00 | -13.00 | 0.00 | 0.00 | 0.00
0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00
--------------------------------------------------------------------------------
Agent at state (4, 2)
0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00
0.00 | -253.00 | -223.00 | -187.00 | -179.00 | -163.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00
0.00 | 3.00 | -241.00 | 0.00 | -135.00 | -115.00 | 0.00 | 0.00 | -1.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00
0.00 | 400.00 | -239.00 | 0.00 | -115.00 | -104.00 | 0.00 | -10.00 | -9.00 | -5.00 | -3.00 | 0.00 | 0.00 | 0.00
0.00 | -66.00 | 0.00 | 0.00 | -102.00 | -94.00 | -58.00 | -46.00 | -34.00 | -22.00 | -13.00 | 0.00 | 0.00 | 0.00
0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Agent at state (5, 12)
0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00
0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00
0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00
0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00
0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00
0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00
--------------------------------------------------------------------------------
Agent at state (5, 13)
0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00
0.00 | -103.00 | -53.00 | -63.00 | -43.00 | -82.00 | 0.00 | 0.00 | -10.00 | -20.00 | -18.00 | 0.00 | 0.00 | 0.00
0.00 | -119.00 | -80.00 | 0.00 | -33.00 | -39.00 | 0.00 | 0.00 | -22.00 | -14.00 | -20.00 | -19.00 | 0.00 | 0.00
0.00 | -106.00 | -108.00 | 0.00 | -58.00 | -39.00 | 0.00 | -34.00 | -27.00 | -21.00 | -25.00 | 0.00 | 0.00 | 0.00
0.00 | -126.00 | -144.00 | 0.00 | -49.00 | -26.00 | -30.00 | -35.00 | -53.00 | -54.00 | -51.00 | 0.00 | 0.00 | 0.00
0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00
--------------------------------------------------------------------------------
```

Figure 7: V-table at various states

Below are explanations for some specific states:

- **State agent (5, 12)**:
  - All values in the table are 0.00.
  - Possible reasons:
    (a) It's a terminal condition, ending the agent's episode.
    (b) The agent hasn't learned a value for actions from this state, perhaps due to never visiting it during training.
    (c) The predicted cumulative reward from this state is 0, regardless of actions.
- **State agent (5, 13)**:
  - This state's values vary from positive to negative to zero.
  - Interpretations:
    (a) Negative numbers suggest certain actions lead to less beneficial outcomes, possibly due to penalties or inefficient paths.
    (b) Zero values could indicate unexplored actions or neutral outcomes.
    (c) In state (5, 13), actions leading to higher (or less negative) V-values are preferred.

In essence, the V-table guides the agent's decisions. Based on V-values of prospective states resulting from various actions, the agent typically chooses the action leading to the highest V-value.

5.2 Plot the learning progress: episode rewards vs. episodes

1. The graph below represents the **Smoothed Learning Progress** of an agent undergoing training in a reinforcement learning environment. It plots the **Episode Rewards** against the number of **Episodes**.

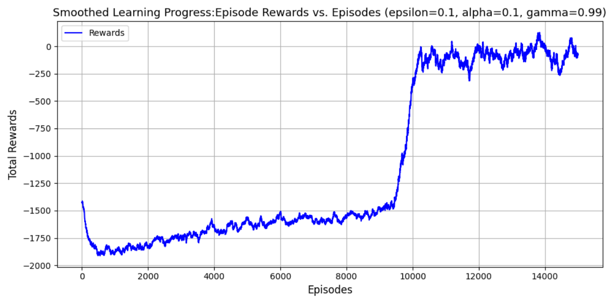


Figure 8: Learning progress of state 1

2. Study Description: "Smoothed Learning Progress: Episode Rewards vs. Episodes (epsilon=0.1, alpha=0.1, gamma=0.99)"

   The graph provides a smoothed representation (likely averaged over several episodes) of the rewards the agent obtains throughout its training. The epsilon, alpha, and gamma parameters are hyperparameters associated with the Q-learning algorithm.

X-axis - ”Episodes”: Represents the number of episodes or iterations the agent has visited in its training journal. As we move rightward, we’re progressing through the agent’s training timeline.

Y-axis - ”Total Rewards”: Represents the total number of rewards accumulated by the agent in each episode. Higher values indicate better performance, while lower values indicate poorer or bad performance.

The Curve (also known as ”Rewards”):

Initial Phase (about 2000 episodes): The agent begins with a very low reward, indicating that it is likely making numerous bad decisions. This is to be expected as the agent explores its environment and learns from its mistakes.

Rapid Improvement Phase (between 2000 and 6000 episodes): The agent’s overall reward significantly increases. This indicates that the agent is learning about the environment and making better decisions. It is leveraging its prior experiences to optimize its rewards.

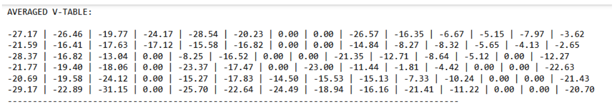
Plateau Phase (After 6000 episodes): Following the quick improvement, the reward stabilizes and hangs around a set value, with slight fluctuations. This indicates that the agent has basically acquired an effective strategy for the environment but still has some room for minor improvements or is exploring other potential options.

The hyperparameters (epsilon=0.1, alpha=0.1, gamma=0.99) are as follows:

The exploration rate is represented by epsilon. A value of 0.1 indicates that the agent chooses a random action (exploration) 10% of the time and the best action (exploitation) 90% of the time. The learning rate, alpha, defines how much of the new Q-value estimate we use in the Q-learning update. The discount factor, gamma, quantifies the agent’s regard for future rewards. A score close to one, such as 0.99, indicates that the agent values future benefits nearly as much as immediate ones.

5.3 Final V-table (shown in the gird) and the policy

1. **Final Averaged V-table (shown in the grid):** The Averaged V-table, shown in this grid style, shows the predicted cumulative rewards (or value) that an agent can expect while starting from each unique condition and following an optimal policy. Each cell in this grid symbolizes a different state. The numerical number within a cell represents the average value of that state, as determined by many occurrences of that state throughout the learning process. Higher-valued states are better for the agent since they indicate higher expected benefits. States with lower or more negative values, on the other hand, are less attractive since they suggest lesser expected rewards or potential penalties. Some cells have a value of ”0.00,” implying that these states were not commonly accessed or modified during the training cycles. figure below shows Final V-table of our study:



```
AVERAGED V-TABLE:

-27.17 | -26.46 | -19.77 | -24.17 | -28.54 | -20.23 | 0.00 | 0.00 | -26.57 | -16.35 | -6.67 | -5.15 | -7.97 | -3.62
-21.59 | -16.41 | -17.63 | -17.12 | -15.58 | -16.82 | 0.00 | 0.00 | -14.84 | -8.27 | -8.32 | -5.65 | -4.13 | -2.65
-28.37 | -16.82 | -13.04 | 0.00 | -8.25 | -16.52 | 0.00 | 0.00 | -21.35 | -12.71 | -8.64 | -5.12 | 0.00 | -12.27
-21.77 | -19.40 | -18.06 | 0.00 | -23.37 | -17.47 | 0.00 | -23.00 | -11.44 | -1.81 | -4.42 | 0.00 | 0.00 | -22.63
-20.69 | -19.58 | -24.12 | 0.00 | -15.27 | -17.83 | -14.50 | -15.53 | -15.13 | -7.33 | -10.24 | 0.00 | 0.00 | -21.43
-29.17 | -22.89 | -31.15 | 0.00 | -25.70 | -22.64 | -24.49 | -18.94 | -16.16 | -21.41 | -11.22 | 0.00 | 0.00 | -20.70
---------------------------------------------------------------------------
```

Figure 9: Learning progress of state 1

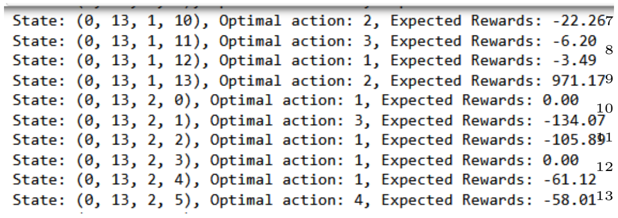
5.4 Learned Policy

formula: **Figure below shows Learned Policy of our study:** The policy specifies the

$$\tilde{Q}_{\pi}(s,a) = \frac{1}{N} \Sigma_{i=1}^{N} G_{i,s}$$

Figure 10: Policy formula [1].

appropriate action an agent should perform in a given condition in order to maximize its expected reward. The learnt Q-values for each state-action pair used to calculate this. **Figure below shows Learned Policy of our study:** To get the largest possible reward, the agent should select the "Optimal action" for each state provided. The "Expected Rewards" field indicates the agent's expected return if the requested action performed in the appropriate state. For a state (0, 13, 1, 13), the agent should select action 2 and expect a payout of 971.17.

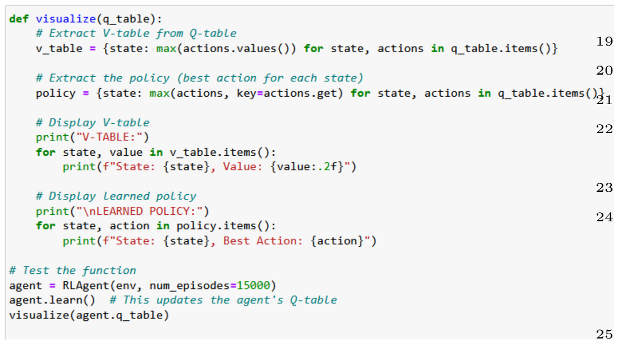


```
State: (0, 13, 1, 10), Optimal action: 2, Expected Rewards: -22.26
State: (0, 13, 1, 11), Optimal action: 3, Expected Rewards: -6.20
State: (0, 13, 1, 12), Optimal action: 1, Expected Rewards: -3.49
State: (0, 13, 1, 13), Optimal action: 2, Expected Rewards: 971.17
State: (0, 13, 2, 0), Optimal action: 1, Expected Rewards: 0.00
State: (0, 13, 2, 1), Optimal action: 3, Expected Rewards: -134.07
State: (0, 13, 2, 2), Optimal action: 1, Expected Rewards: -105.89
State: (0, 13, 2, 3), Optimal action: 1, Expected Rewards: 0.00
State: (0, 13, 2, 4), Optimal action: 1, Expected Rewards: -61.12
State: (0, 13, 2, 5), Optimal action: 4, Expected Rewards: -58.01
```

Figure 11: Learned Policy

5.5 Modified Code for Visualize Method

The code were modied for the purpose of displaying V-tables and Learned policy under 15000 episodes.



```python
def visualize(q_table):
    # Extract V-table from Q-table
    v_table = {state: max(actions.values()) for state, actions in q_table.items()}

    # Extract the policy (best action for each state)
    policy = {state: max(actions, key=actions.get) for state, actions in q_table.items()}

    # Display V-table
    print("V-TABLE:")
    for state, value in v_table.items():
        print(f"State: {state}, Value: {value:.2f}")

    # Display learned policy
    print("\nLEARNED POLICY:")
    for state, action in policy.items():
        print(f"State: {state}, Best Action: {action}")

# Test the function
agent = RLAgent(env, num_episodes=15000)
agent.learn()  # This updates the agent's Q-table
visualize(agent.q_table)
```

Figure 12: Visualization Method

6 Additional Task

I handled all errors that appeared in Game Interface, where now you can play game manually without any error but interacting with only restrictions (Settings). **Snipped codes for Interface**

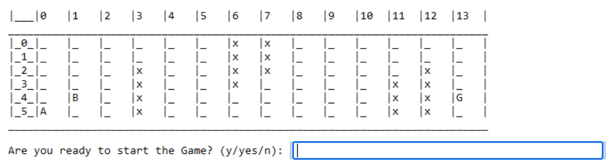
```python
env = CliffBoxGridWorld(render=True)
env.reset()
env.print_world()

ready = input("Are you ready to start the Game? (y/yes/n): ").lower()

if ready == 'y' or ready == 'yes':
    done = False
    rewards = []

    while not done:
        try:
            print("Any mistake will cause zero tolerance! \n")
            action = int(input("Please input the actions (up: 1, down: 2, left: 3, right: 4) or 0 to exit Game: "))
            if action == 0:
                print("Game's closed!")
                break
            elif action not in [1,2,3,4]:
                print("Invalid Action.")
                print("Game Declined.")
                break
            state, reward, done, info = env.step(action)
            rewards.append(reward)
            print(f'step: {env.timesteps}, state: {state}, actions: {action}, reward: {reward}')
            env.print_world()
            print(f'rewards: {sum(rewards)}')
            print(f'action history: {env.action_history}')
        except ValueError:
            print("Internal Error, Try again!")
            break
else:
    print("Okay,get prepaire and come
        ↪ back when you're ready!")
```

First Interface before Game start Manually



```
|___|0   |1   |2   |3   |4   |5   |6   |7   |8   |9   |10  |11  |12  |13  |
__________________________________________________________________________
|_0_|_   |_   |_   |_   |_   |_   |x   |x   |_   |_   |_   |_   |_   |_   |
|_1_|_   |_   |_   |_   |_   |_   |x   |x   |_   |_   |_   |_   |_   |_   |
|_2_|_   |_   |_   |x   |_   |_   |x   |x   |_   |_   |_   |_   |x   |_   |
|_3_|_   |_   |_   |x   |_   |_   |x   |_   |_   |_   |_   |x   |x   |_   |
|_4_|_   |B   |_   |x   |_   |_   |_   |_   |_   |_   |_   |x   |x   |G   |
|_5_|A   |_   |_   |x   |_   |_   |_   |_   |_   |_   |_   |x   |x   |_   |
__________________________________________________________________________
Are you ready to start the Game? (y/yes/n): |
```

Figure 13: Manual Game Start with free errors

7 Conclusion

The graph depicts how an agent's learning evolves over time. The agent performs poorly at first but quickly improves as it acquires experience. The agent's performance eventually stabilizes, showing that it has learnt how to traverse its environment in order to maximize its rewards given the provided hyperparameters. Note: Game played dynamically or manually.

References

[1] Assoc Prof Bo AN. Reinforcement learning, 2023. Accessed: 2023.

[2] A. Author and B. Coauthor. Chatgpt: Conversational models in action. *Journal of Artificial Intelligence*, 123(4):456–789, 2023.

[3] DataCamp. Introduction to q-learning: Beginner tutorial, 2022.