Kevin Ferreira
CAP 2269

Reinforcement learning is where an agent gains experience by interacting with the environment to maximize rewards. This project will demonstrate reinforcement learning with a grid world problem where an agent must navigate from a start to end while avoiding obstacles. Two different algorithms will be used for this project which are Q-learning and SASRA. According to Watkins (1992), q-learning is a reinforcement learning model where an agent acts optimally in Markovian environments which means that the agent learns optimal actions by updating estimates and SASRA is where the estimates update based on the state, action, reward, next state, and the next action actually taken. Overall, this project will study reinforcement learning via the grid world problem which is motivated by uses such as navigation planning and robotics.

The project takes place in a 10x10 grid world with walls. The agent must move from a starting point to the goal and the agent can only move up, down, left, and right. The agent will stay the same if it attempts to move into a wall or the border of the grid world. The reward function is that the agent loses a point every time it moves and gains 100 points when it reaches the goal. With these conditions, the agent will be motivated to find the shortest path possible. After the grid world was created, two algorithms were implemented which were Q-learning and SASRA which the expected solution is to find the best paths to the goal. The parameters set for both algorithms were 5000 iterations, discount factor at 0.9, learning rate at 0.1, and exploration rate at 0.9. Once the experiment is run, steps-to-goal curves will be visualized and the sum of squared errors and the theoretical value table will be produced.

Everything in this project was implemented in Python. In the first cell, the code sets up the grid world and parameters and the q-table for the algorithms. In the second cell, the code defines three functions that are essential to the project which are “is_valid_state” which determines if a position to move to is valid or not, “take_action” which makes the agent move if the next state is valid and the “choose_action” defines the e-greedy policy for action selection. In the third and fourth cell, the code defines the function for q-learning by “new_q_value = old_q_value + alpha * (reward + discount * max_future_q - old_q_value)” and SASRA by “new_q_value = old_q_value + alpha * (reward + discount * future_q_value - old_q_value)” respectively. In the fifth and sixth cell, they plot the steps to goal curves for each algorithm. In the seventh cell, the code defines the function to write the theoretical value table. In the eighth cell, the code calculates the sum of squared errors.

Overall, this project tests two algorithms which are Q-learning and SASRA in a 10x10 grid where an agent must move from start to end while avoiding walls. Steps to goal curves, a theoretical value table and sum of squared errors were also obtained.

From the results, both algorithms were able to develop near-optimal paths and they have the same greedy policies. Q-learning has a lower sum of squared error compared to SASRA which means that its q-values are closer to theoretical optimal values. The steps to goal curves also show that early episodes have larger variances but become smaller in later episodes in both algorithms. Overall, this project proves how effective both algorithms are and how they compare.

Reference:

Watkins, Christopher J. C. H., and Peter Dayan. “Q-Learning - Machine Learning.” *SpringerLink*, Kluwer Academic Publishers, link.springer.com/article/10.1007/BF00992698. Accessed 25 Nov. 2025.

Appendix:

Project Link:
https://colab.research.google.com/drive/1-QxCHCvlohNyfUUfX2cUwjRTX-RkRZH-?usp=sharing

Pseudocode:
Q-learning:

```
INITIALIZE Q[r, c, a] = 0 for all states (r, c) and actions a
FOR episode = 1 TO N_EPISODES:
   state = START_STATE
   steps = 0
   WHILE state != GOAL AND steps < MAX_STEPS:
      steps = steps + 1
      WITH probability ε:
         action = random action from A
      ELSE:
         action = argmax_a Q[state, a]
      next_state, reward = take_action(state, action)
      best_next = max_a' Q[next_state, a']
      Q[state, action] = Q[state, action] +
                  α * (reward + γ * best_next - Q[state, action])
      state = next_state
END FOR
```

SASRA:
INITIALIZE Q[r, c, a] = 0 for all states (r, c) and actions a
FOR episode = 1 TO N_EPISODES:
    state = START_STATE
    WITH probability ε:
        action = random action
    ELSE:
        action = argmax_a Q[state, a]
    steps = 0
    WHILE state != GOAL AND steps < MAX_STEPS:
        steps = steps + 1
        next_state, reward = take_action(state, action)
        WITH probability ε:
            next_action = random action
        ELSE:
            next_action = argmax_a Q[next_state, a]
        Q[state, action] = Q[state, action] +
                        α * (reward + γ * Q[next_state, next_action]
                            - Q[state, action])

        state = next_state
        action = next_action
END FOR

Input Sequence:
(st,at,rt+1,st+1)

Output Sequence:
argmaxQ(s,a),