1 Problem 1

1.1 Problem Analysis

Our task was to implement value iteration agent. Value iteration is used to solve Markov Decision Processes (MDPs) to determine the optimal policy that maximizes the collective reward for an agent over time.

1.2 Solution

The agent is initialized with an MDP, a number of iterations, and a discount factor. The estimated state values are computed by using runValueIteration after initializing a values dictionary with those values.
The runValueIteration method determines the maximum Q-value and calculates the Q-values for all the possible actions in order to update the value for each state in each iteration. To maintain consistency, we use the values from the previous iteration.
To compute the Q-values, the computeQValueFromValues method is used. The Q-value for a state-action pair is determined by adding the reward and the discounted value of the next state, and summing the expected utility of transitioning to every potential next state, weighted by their transition probabilities.
The computeActionFromValues method chooses the most optimal action to take. If there are no more options left, it indicates that a state is terminal. For terminal states, it returns None. Otherwise, it calculates the Q-values for each potential action and chooses the one with the highest Q-value.

1.3 Findings

The implementation of value iteration combines iterative updates, Q-value computation and optimal action selection to solve an MDP. It ensures that the agent maximizes its expected cumulative reward.

2 Problem 2

2.1 Problem Analysis

Our task was to change one of the two parameters noise or discount to make the agent cross the bridge.

2.2 Solution

In the BridgeGrid environment, the default value of parameter noise was 0.2. So the agent was avoiding crossing the bridge because of the high risk of falling into the chasm. I set the noise to 0.0 to eliminates the risk of fallling into the gap and getting negative rewards.
I also set the value of the parameter answerDiscount to 0.9 to encourage the agent to cross the bridge and collect more rewards.

2.3 Findings

By reducing the value of the noise and increasing the value of discount, it is possible to achieve the intended behavior of the agent.

3 Problem 3

3.1 Problem Analysis

Our task was to task try out different value of three parameters - noise, discount and the living reward in an MDP to produce different optimal policy types.

3.2 Solution

3.2.1 3a

A low value such as 0.3 is chosen for answerdiscount parameter to encourage the agent to take immediate rewards instead of future rewards. A living reward of set to 0 to indicate that there is no reward for living longer.

3.2.2 3b

answerNoise is set to 0.2 along with the previous values of other two parameters to achieve the required policy.

3.2.3 3c

The answerDiscount is set to 0.9 to encourage the agent to go for the distant exit with higher reward. The rest of the parameters are set to 0.

3.2.4 3d

answerNoise is set to 0.2.

3.2.5 3e

The answerDiscount and answerNoise is set to 0 and answerLivingReward is set to 1 to ensure that the agent keeps moving indefinitely without reaching the goal.

3.3 Findings

Different values for different parameters enforces unique behaviours of the agent.

4 Problem 4

4.1 Problem Analysis

Our task was to write a value iteration agent AsynchronousValueIterationAgent. The agent updates only one state in each iteration.

4.2 Solution

The AsynchronousValueIterationAgent implements value iteration in an asynchronous manner. In each iteration, it updates the value of a single state rather than all states. Specifically, it updates one state per iteration based on the maximum Q-value of its possible actions.
The Q-values are computed by considering the immediate reward and the discounted value of subsequent states which reflects the expected utility of transitioning to potential next states.

This approach ensures the value function gradually converges to the optimal value by updating the value of each state iteratively.

4.3 Findings

This method can lead to faster convergence in large MDPs as it provides benefits in terms of computational efficiency.

5 Problem 5

5.1 Problem Analysis

Our task was to implement PrioritizedSweepingValueIterationAgent.

5.2 Solution

The PrioritizedSweepingValueIterationAgent focuses on updating state values that are most likely to change the policy, using a prioritized sweeping algorithm. At first we identify the predecessors for each state, which are states that can transition to the given state through some action.
Then we use a priority queue to manage state updates based on their priority, which is determined by the difference between the current value of the state and the highest Q-value across all possible actions.
For each non-terminal state, we calculate the difference and push the state into the priority queue with priority -diff.
In each iteration, we pop the state with the highest priority (largest diff) from the queue, update the state's value and for each predecessor of the state, we calculate the new diff and push it into the priority queue if the diff exceeds a threshold.
This approach ensures that states with the highest potential impact on the policy are updated first, leading to more efficient convergence.

—

5.3 Findings

By leveraging a priority queue and the concept of predecessors, the algorithm efficiently directs towards the most impactful updates. It leads to faster and more efficient convergence to the optimal policy. This method is particularly beneficial in large or complex state spaces where traditional value iteration may be computationally difficult.

6 Challenges

It took me some time to understand the impact of different parameters in task 2 and 3. I tried out multiple combinations of value to understand it thoroughly which consumed most of my time during the lab session.