Islamic University of Technology

Report on Lab 05

(CSE 4618 Artificial Intelligence Lab)

Name:	**Md Mahmudul Islam Mahin**
Student ID:	**210042140**
Department:	CSE
Program:	BSc in SWE
Date:	1st September 2025

Contents

1 Introduction

In this lab, we will implement value iteration. We will test our agent on Gridworld.

2 Question 01: Value Iteration

To define runValueIteration( ) we need to define some assisting functions –

2.1 ComputeQValueFromValues

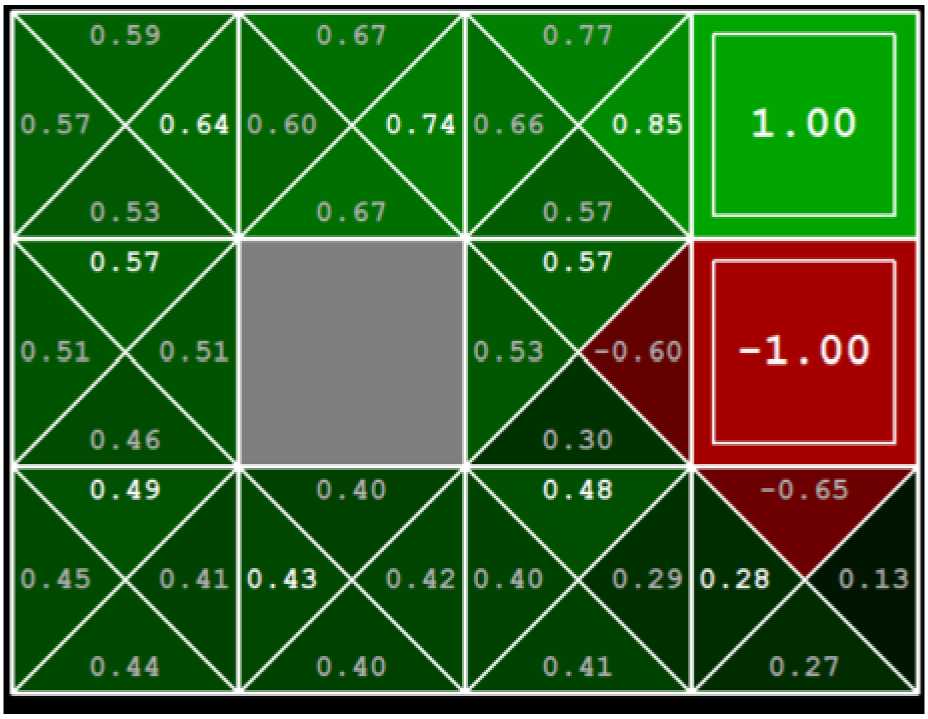
The function computes the Q-value for a specific state-action pair. The Q-value is determined by summing the Living Reward and the discounted reward from the previous state, each multiplied by the Transition probability, considering all potential actions from the current state.

2.2 ComputeActionFromValues

The function iterates through all possible actions for a given state, calculates their respective Q-values using the getQValue method, and then identifies and returns the action with the highest Q-value. Here Counter is a data structure from utility which returns 0 when it's empty .

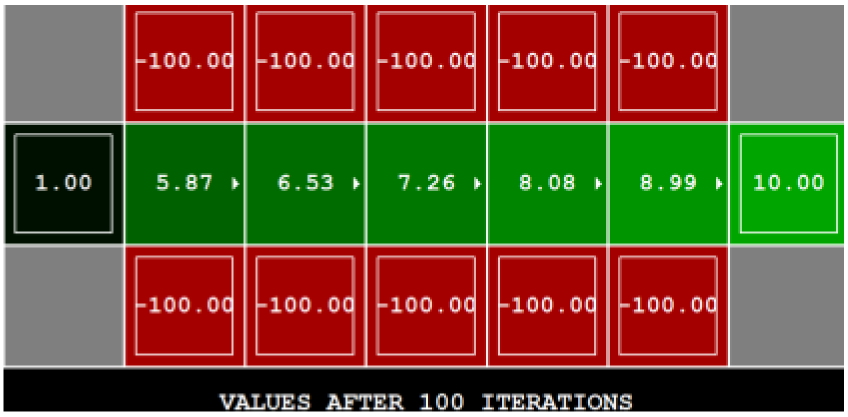
2.3 runValueIteration

The code performs a number of iterations, and in each iteration, it updates the value of each state based on the maximum Q-value calculated for the available actions in that state. The process involves creating a copy of the current state values (so that the original current state values are unaltered), calculating Q-values, and updating the values with the maximum Q-value for each state. This helps refine the estimated values of states over multiple iterations. This function performs a batch update of the values of all the states in each iteration.



3 Question 02: Bridge Crossing Analysis

Initially the Discount factor was set and noise was set as 0.9 and 0.2 respectively. Higher discount factor means we are prioritizing distant reward over closer reward, and noise value denotes how often an agent ends up in an unintended position while executing an action. As we can change only one value, we chose to significantly reduce noise (closer to zero) as we cant risk falling off the cliff.

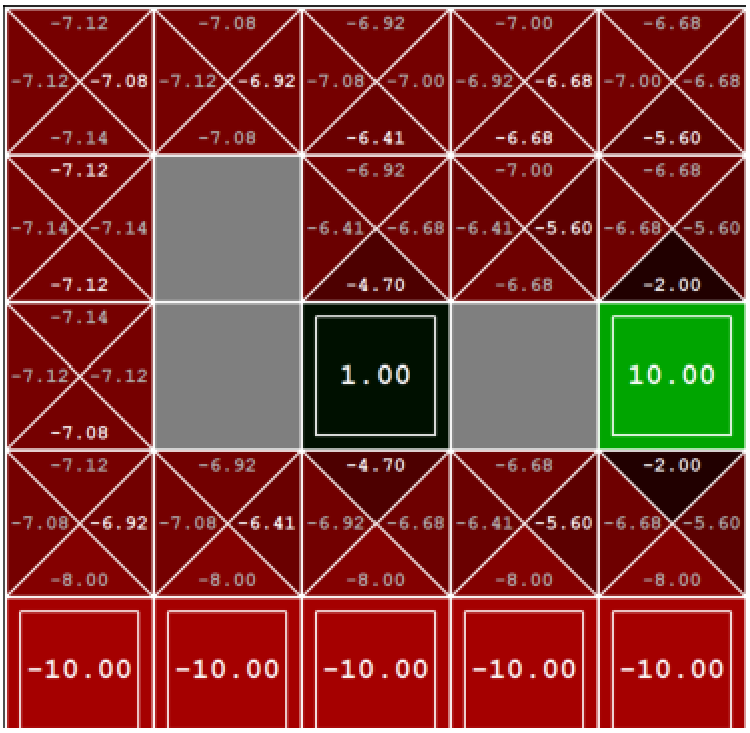


4 Question 03: Policies

The optimal policy types that should be attempted to produce:
a. Prefer the close exit (+1), risking the cliff (-10)
b. Prefer the close exit (+1), avoiding the cliff (-1)
c. Prefer the distant exit (+10), risking the cliff (-10)
d. Prefer the distant exit (+10), avoiding the cliff (-10)
e. Avoid both exits and the cliff (so an episode should never terminate)

4.1 Prefer the close exit (+1), risking the cliff (-10)

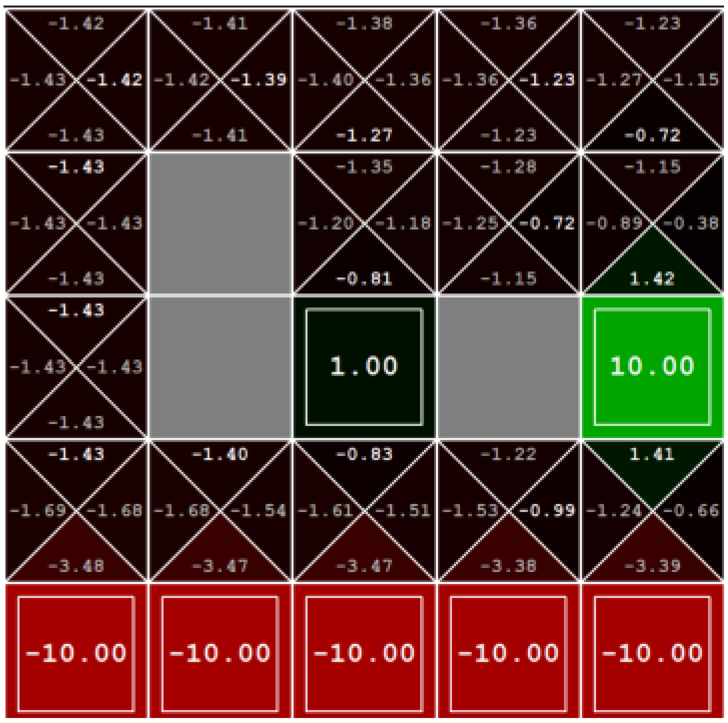
The discount factor was set low so that the agent prioritizes closer exit (+1). The noise value was set as 0 as we didn't wanted to risk falling off the cliff while taking the path closer to the cliff. And the living reward was set low so that it chooses to find the closer exit as living is expensive (high negative reward).



Even though the qValues or the state values are negative, they lead to the closest exit (+1) risking the cliff from the starting point.

4.2 Prefer the close exit (+1), avoiding the cliff (-1)

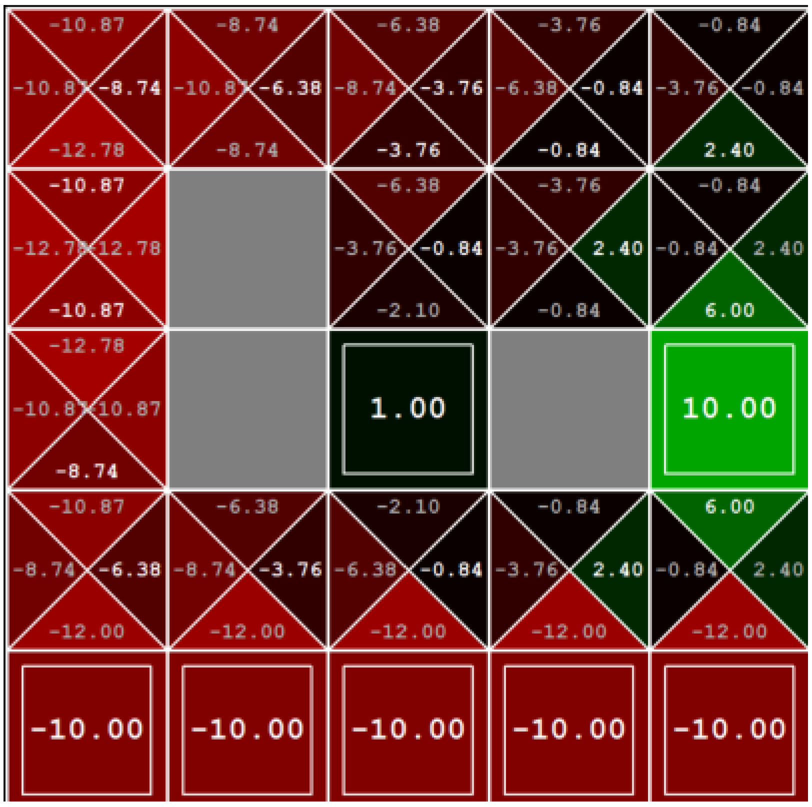
The discount factor was set low so that the agent prioritizes closer exit (+1). The noise value was set as 0.2, here we allowed a bit of noise as its taking a safer route than before where there's less risk of falling off a cliff. The living reward was set higher than before as this time its taking the lengthier route to the closest exit.



This time the values increase a bit due to increase in living reward and it leads to the closest exit (+1) avoiding the cliff.

4.3 Prefer the distant exit (+10), risking the cliff (-10)

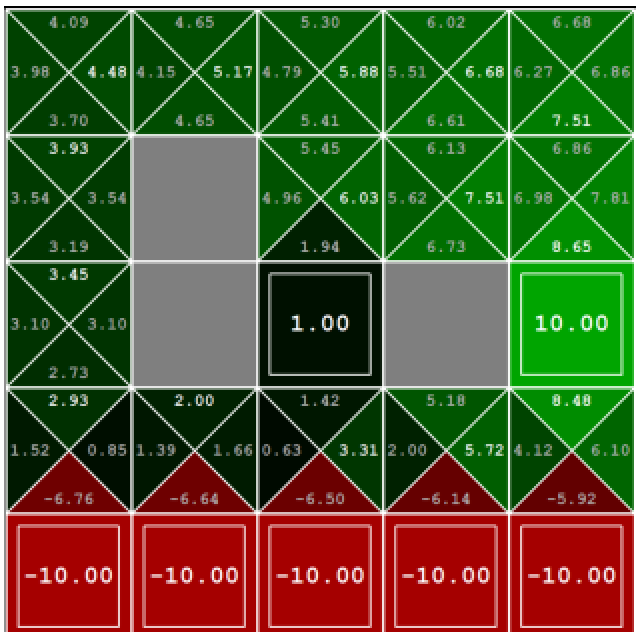
The discount factor was set high so that the agent prioritizes distant exit (+10). The noise value was set as 0 as we didn't want to risk falling off the cliff while taking the path closer to the cliff. The living reward was set in such a way that it wants to exit soon but it ends up in the distant exit rather than the closer one.



This time the values decrease again due to negative living reward and it leads to the distant exit (+10) risking the cliff.

4.4 Prefer the distant exit (+10), avoiding the cliff (-10)

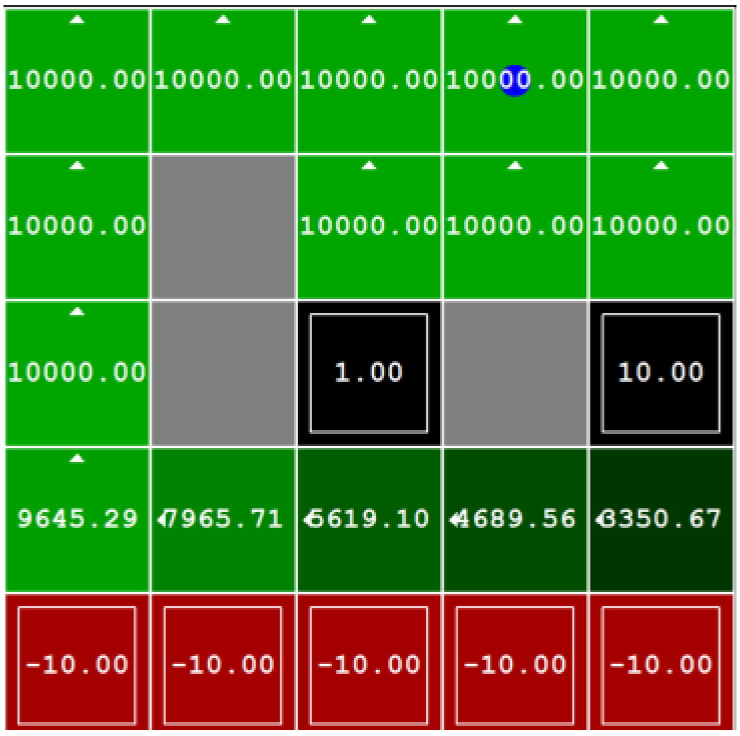
The discount factor was set high so that the agent prioritizes distant exit (+10). The noise value was set as 0.2, here we allowed a bit of noise as it's taking a safer route than before where there's less risk of falling off a cliff. The living reward was set higher than before as this time its taking the lengthier route to the farthest exit. So it can afford to remain in the grid longer and exit through the distant exit.



The values looks much better this time due to non negative living reward, and it leads to the distant reward (+10) avoiding the cliff.

4.5 Avoid both exits and the cliff (so an episode should never terminate)

The goal of the agent is to never terminate, so we set a very high living reward value so that it keeps looping without exiting.



The living reward being too high, it loops within the grid and tries to remain alive.

5 Question 04: Asynchronous value Iteration

The solution approach is quite similar to that of (synchronous) value iteration but here we maintain a queue of states and in every iteration we update the state value of a single state which is the top element of the queue. The updated state in each iteration is removed from the queue and pushed again into the end of the queue. In synchronous value iteration, the state value update of every state is done in each iteration as a batch update, whereas in this case a single state is updated per iteration.

6 Question 05: Prioritized Sweeping Value Iteration

The solution approach is similar to AsynchronousValueIterationAgent with some minor changes. This time we update the current state values of the states based on a priority rather than randomly. We can divide the code into the following segments

6.1 Compute the predecessors

It iterates through all states in the MDP that are not terminal states. For each state, it considers all possible actions and their associated next states with transition probabilities. It then updates the predecessors dictionary, associating each next state with a set of predecessor states.

6.2 Compute the priority

It iterates through non-terminal states in the MDP, calculates the Q-values for each possible action in a state, and computes the difference between the maximum Q-value and the current value for that state. The states are then added to the priority queue based on the negative of this difference, intending to prioritize states with larger differences first. The negative sign is used to prioritize maximum difference in a mean heap based priority queue.

6.3 Updating state value of current state and its predecessors

The code iteratively updates non terminal state values based on the maximum Q-values, prioritizing states with larger differences. It also updates the predecessor of the current state if the difference of its current state value and previous state value is greater than theta. This algorithm is also a form of asynchronous value iteration, but here it deals with the states with higher difference of current and previous state value to focus on the updates that might be responsible to change the policy.

7 GitHub Repository

I will be uploading the lab tasks in the following repository: CSE 4618: Artificial Intelligence