CS 4320/5314

Programming Assignment 3:

Solving MDPs

Group Members:

Menda W Dorji

Sonam S Tshering

**Contributions**

Menda Wangchuk Dorji – Q-Learning, Value Iteration.

Sonam Seldon Tshering – Monte Carlo, Value Iteration.

Both of us contributed to group meetings and collectively worked on the code together. We both ran and tested the code. Both of us contributed to documentation, testing, and report writing.

Note: the functions were assigned individually but we both helped equally on each other’s part and gave review as well.

**Introduction**

In this assignment we developed AI agents for MDP-based simulation of a student’s life, where the students can find themselves in different states, representing their mental and academic condition. The objective is to learn optimal decision-making strategies to balance a good time with academic success.

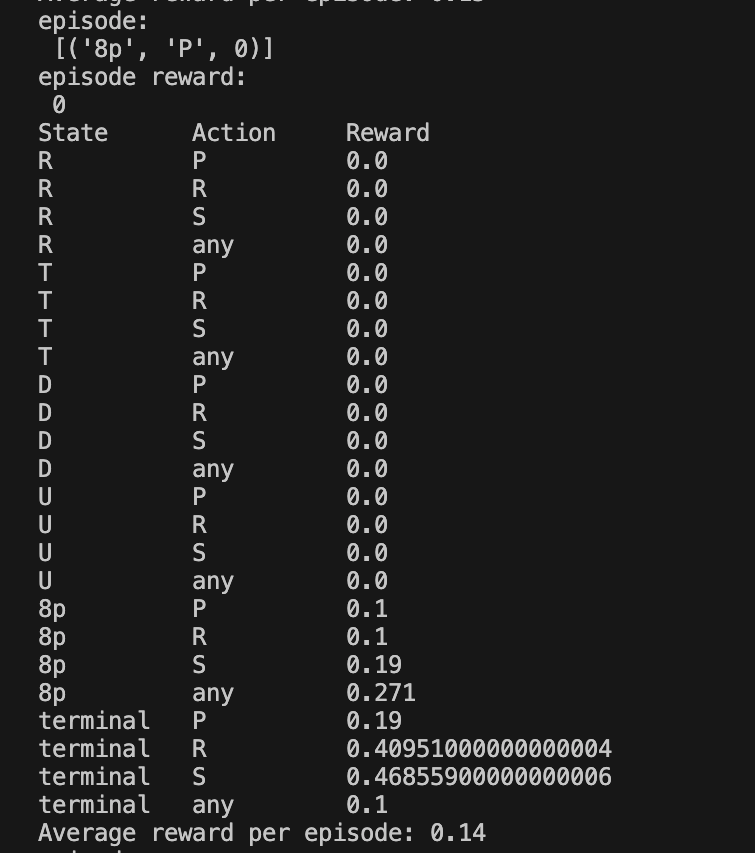
We have a separate result file for each algorithms (please scroll down to view final results).

What we implemented for the three algorithms are given below:

**Algorithm 1: Monte Carlo**

 In this algorithm, we evaluated the state-action values of a Markov Decision Process (MDP) by simulating episodes and updating the values based on the total rewards obtained. In each episode, the algorithm randomly selects actions based on a policy, observes the resulting rewards and states, and then updates the state-action values using a first-visit Monte Carlo update rule. The process is repeated for multiple episodes to converge towards accurate state-action values.

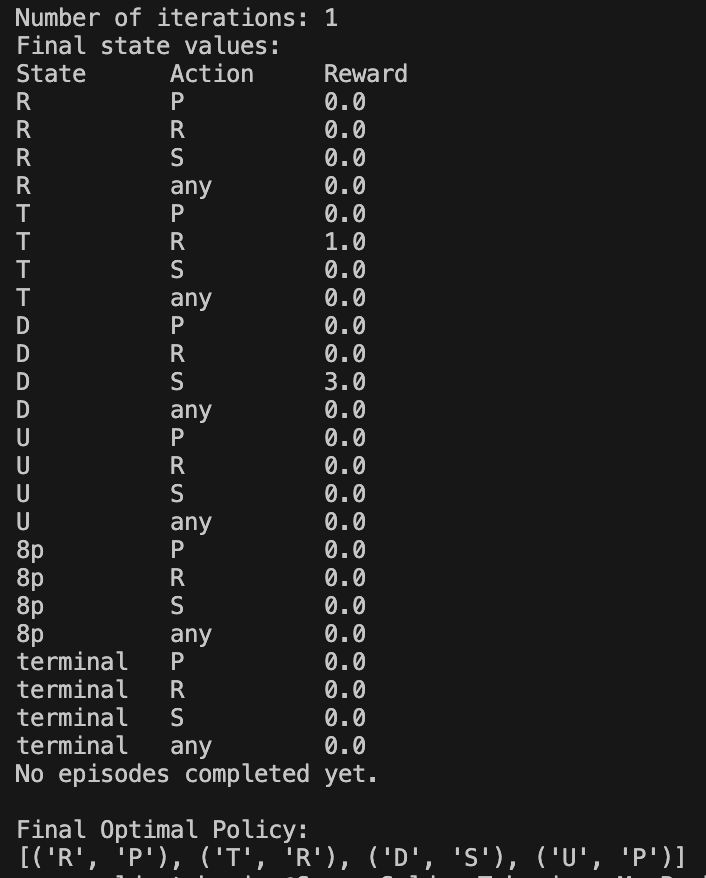
**Results**



**Algorithm 2: Value Iteration**

In this algorithm, we found the optimal policy for an MDP by iteratively improving the estimated state values until they converge. The algorithm iteratively calculates the expected values for all possible actions in each state and updates the state values using the maximum expected value. The process continues until the change in values becomes smaller than a predefined threshold (epsilon). The final result provides the optimal state-action values and the corresponding optimal policy.

**Results**

****

**Algorithm 3: Q-Learning**

In this algorithm, we trained an agent to make optimal decisions in an environment by learning the Q-values, representing the expected cumulative rewards for taking specific actions in specific states. The algorithm starts with an initial Q-table and explores the environment by taking actions, observing rewards, and updating the Q-values accordingly. Q-values are updated using the Q-Learning formula, which considers both immediate rewards and the maximum Q-value for the next state. Over multiple episodes, the Q-table converges to optimal values, and the agent can use this information to make optimal decisions in the given environment.

**Results**

