REINFORCEMENT LEARNING CW1 REPORT – LEILI BAREKATAIN

Question 1:

a) I chose **Policy Iteration** to solve this problem, as it effectively combines **Policy Evaluation** and **Policy Improvement** to find an optimal policy. This method is particularly suitable for small to medium-sized state spaces, where it converges efficiently. For the stopping condition in Policy Evaluation, I set a **threshold of 0.0001** to achieve a balance between accuracy and computational efficiency. A smaller threshold provides more accurate and stable values but may require more iterations. Given the small state space in this case, using this lower threshold is not computationally expensive and provides a reliable solution. Moreover, I **initialized policy to choose action 1 systematically** because it allows the algorithm to iteratively improve toward the optimal policy by evaluating and updating actions across states and it provides a consistent baseline for Policy Iteration.

b)

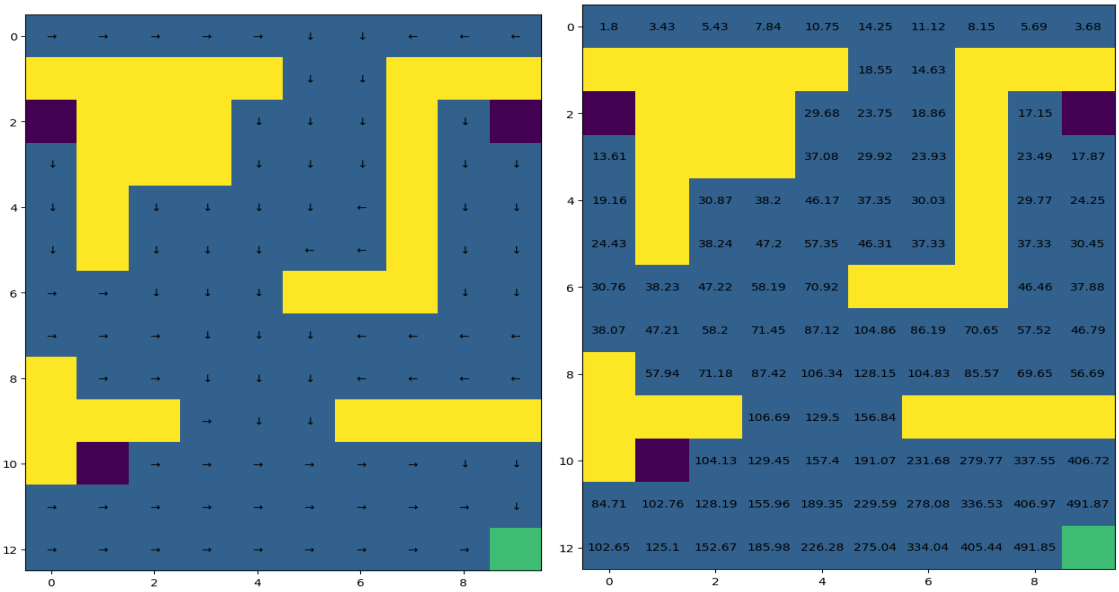

Figure 1: optimal policy and optimal values for DP Agent

c) When rewards are scaled by a factor of 2, **value function V(s) will also scale by the same factor**. This is because values are updated according to this:

$$V(s) = \sum_a \pi(a|s) \sum_{s'} T(s'|s,a) \left[R(s,a,s') + \gamma V(s') \right]$$

Therefore, if each reward doubles, the total expected reward will also double and thus the V(s). However, **the optimal policy will not change**, because value function will be shifted uniformly

across states. According to the policy update rule:

$$\pi(s) = \arg\max_a \sum_{s'} T(s'|s,a)\left[R(s,a,s') + \gamma V(s')\right]$$

we are getting the argmax, so scaling values does not affect which actions are valued more relative to others in each state, and therefore the optimal policy stays the same.

d) To design the reward function so that the value V(s) for any state s represents the probability of eventually reaching the goal state, my choices are as follows:

1. **Discount factor γ = 1**, because we want to give equal importance to our immediate reward and future reward. This aligns with computing the probability of reaching the goal rather than prioritizing immediate rewards.
2. **Reward for steps that don't reach a terminal state = 0**, because our focus is on reaching the goal state which is a terminal state, so we should make sure that non-terminal states contribute nothing to the value function.
3. **Reward for reaching the goal state = 1**, because we want the agent to get reward when reaching the goal state. This encourages the agent to prioritize paths that lead to the goal state.
4. **Reward for reaching other terminal states = 0**, because our focus is on reaching the goal state (not every terminal state). This discourages paths that lead to non-goal terminals, thereby lowering the value for states that may end in non-goal terminals.

With these settings, we ensure that the value of a state will be 1 if starting from that state, we eventually reach the goal state, and it is 0 if the path is likely to lead to a non-terminal state (never reaches the goal) or any other terminal state rather than goal state. Thus, V(s) becomes the probability of reaching the goal starting from state s.

Question 2:

a) I used "**On-policy ε-greedy first-visit Monte Carlo control algorithm**", because it balances exploration and exploitation. I selected the epsilon by plotting MC total rewards over episodes for different epsilon values. As seen in the plot below, **ε=0.4** provides a good balance between exploration and exploitation, resulting in relatively stable total rewards over episodes. I selected the **number of episodes = 2000**, as this duration allows sufficient time for the agent's learning curve to stabilize, as shown in the learning curve plot in part (c). Although convergence occurs before 2000 episodes (around 700), this duration ensures consistent results and reduces variability in rewards, making the learned policy more stable across different runs. I **initialized the policy for Monte Carlo as a uniform distribution, with each action having equal probability** $\frac{\mathbf{1}}{(\boldsymbol{action\ size})}$. This provides an unbiased starting point, allowing the agent to explore all actions equally before refining toward the optimal policy. This approach ensures sufficient exploration in the early stages, helping the Monte Carlo algorithm gather information across all actions for accurate value estimation. I didn't change the initial values of Q and V. Q is initialized randomly as this can encourage exploration at the start of learning, and V is initialized to zeros because this allows the MC to build state values from scratch based on the returns observed over time, without any initial bias.

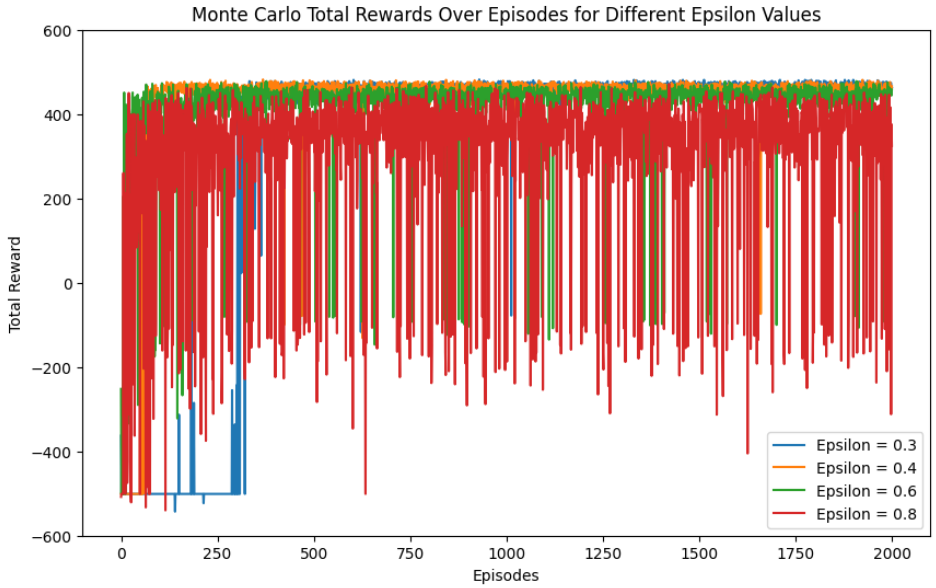

Figure 2: MC Total Rewards Over Episodes for Different Epsilon Values

b)

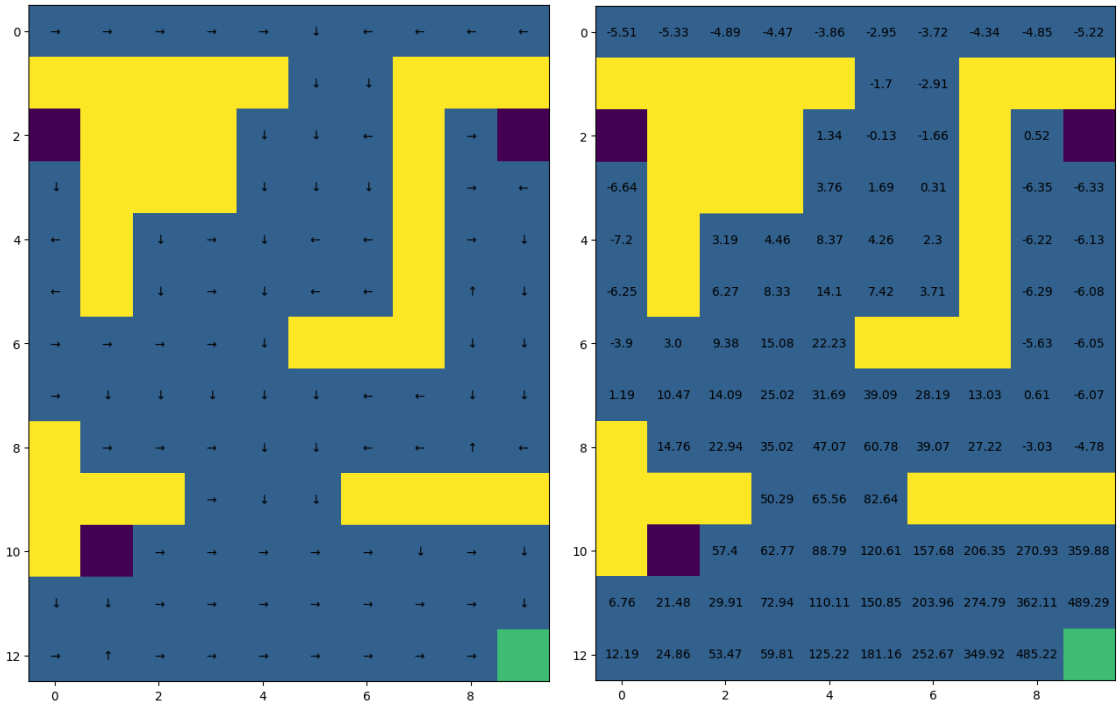

Figure 3: Optimal Policy and Optimal Value for MC Agent

c) **Learning curve:** MC Agent has a relatively high variance because it updates values only after complete episodes, causing fluctuations as it explores different paths before convergence.

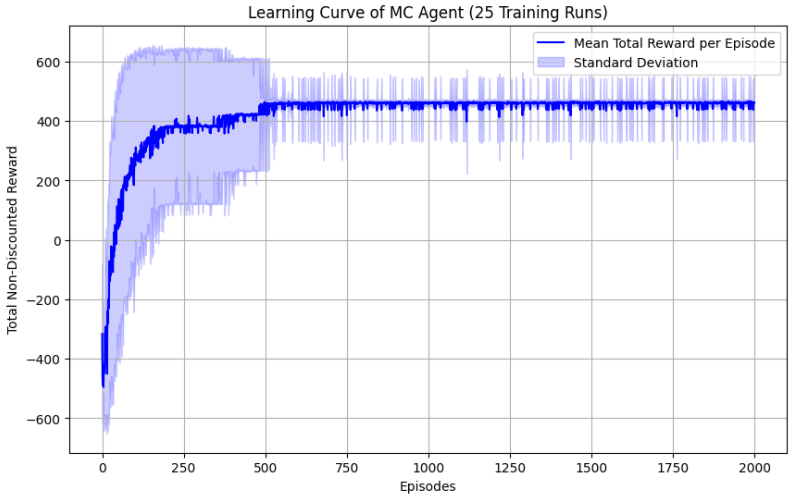

Figure 4: MC Learning Curve

d) If we ignore trajectories that don't reach terminal states within 500 steps, this **limits exploration**, particularly in larger or more complex environments with longer paths to the goal. This step limit **introduces bias** in the value estimates, as only shorter, successful trajectories are used to update the value function. While the step limit **reduces variance for states near terminal states** (which reach terminals more consistently), it can **increase variance for distant states**, where longer trajectories are discarded, resulting in less reliable estimates. Moreover, the agent **may not be able to converge to the optimal values**, since Monte Carlo relies on the average return over complete trajectories to estimate values accurately, therefore removing long paths can prevent the value function from reflecting the actual probabilities of reaching terminal states from all states. On the other hand, the 500-step limit makes the MC algorithm **more computationally efficien**t by constraining episode length. This reduces the computation time needed, especially in large or complex environments where paths might otherwise be very long.

Question 3:

a) I chose **Q-learning** because it is an off-policy TD learning algorithm that allows the agent to explore freely (with an ε-greedy policy) while still converging to the optimal policy, making it well-suited for environments with stochastic transitions and uncertain paths. To select the optimal values for epsilon and learning rate (alpha), I plotted a heatmap showing the average total rewards for different combinations of these parameters. As seen in the heatmap, lower epsilon values combined with moderate alpha values yield the highest rewards, indicating a good balance between exploration and exploitation. Based on this, I chose **ε = 0.2** and **alpha = 0.5** as they provide stable and high rewards, optimizing the agent's performance. Finally, I selected the **number of episodes = 800**, as this duration allows sufficient time for the agent's learning curve to stabilize, as shown in the learning curve plot in part (c). Although convergence occurs before 800 episodes, this duration ensures consistent results and reduces variability in rewards, making the learned policy more stable across different runs. I didn't change the initial values of Q, V, and Policy. Q is initialized with random values which encourages exploration, V is initialized to zeros which provides a neutral state-value baseline, and policy is initialized to zeros which creates a structure for updating action choices based on learned Q-values.

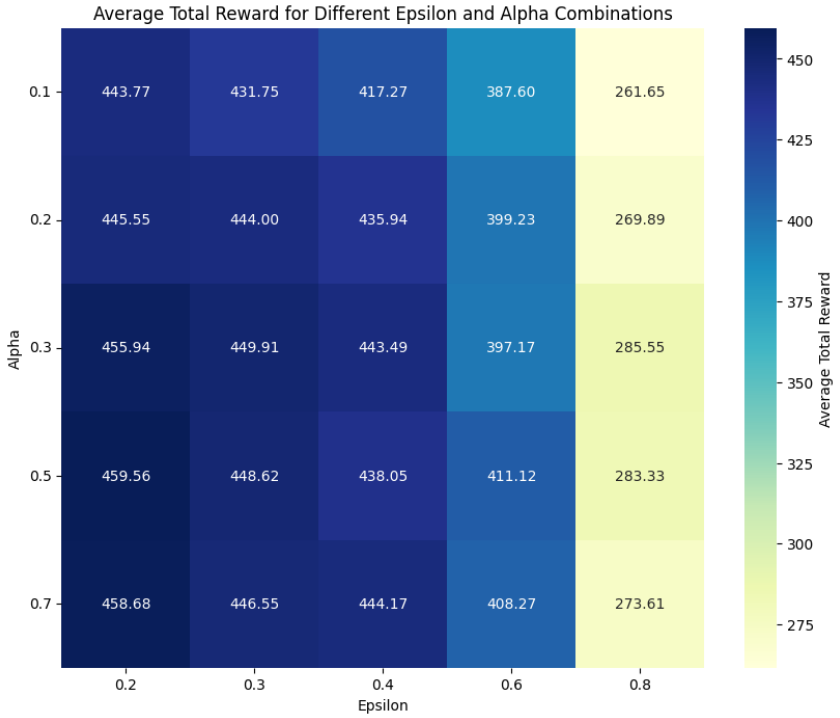

Figure 5: Heatmap of Average Total Rewards for Different Epsilon and Alpha Combinations in TD Learning

b)

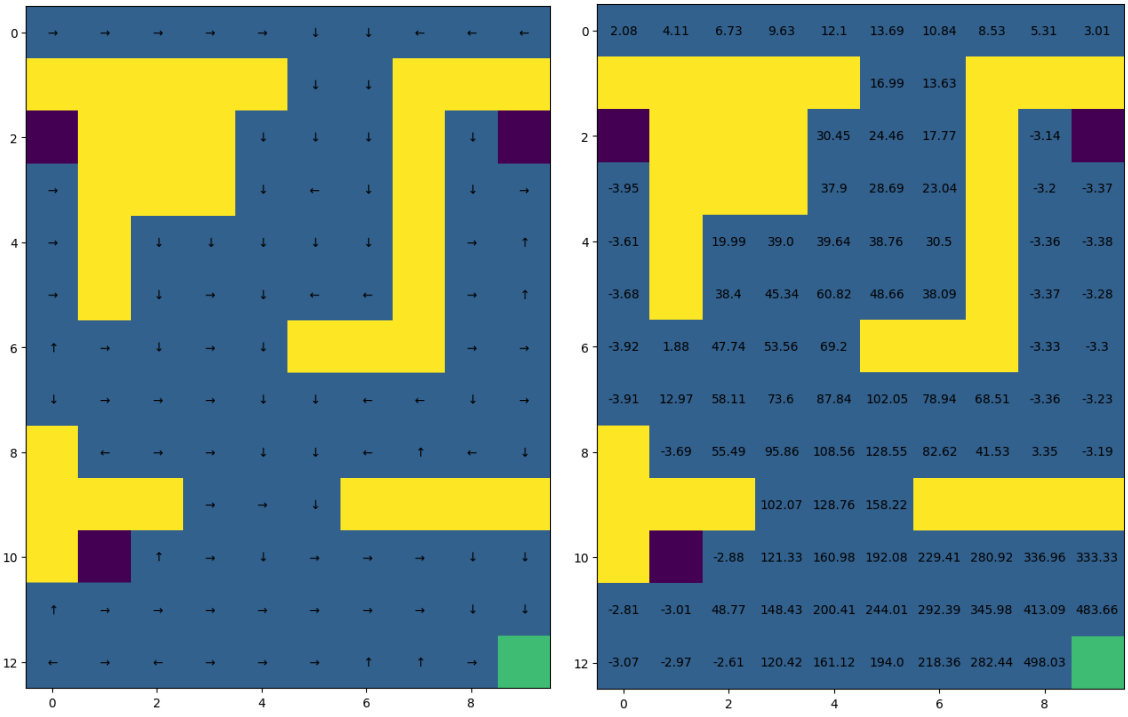

Figure 6: Optimal Policy and Optimal Value for TD Agent

c) **Learning Curve:** TD has lower variance than MC and it converges faster because it updates values incrementally at each step rather than waiting until the end of an episode.

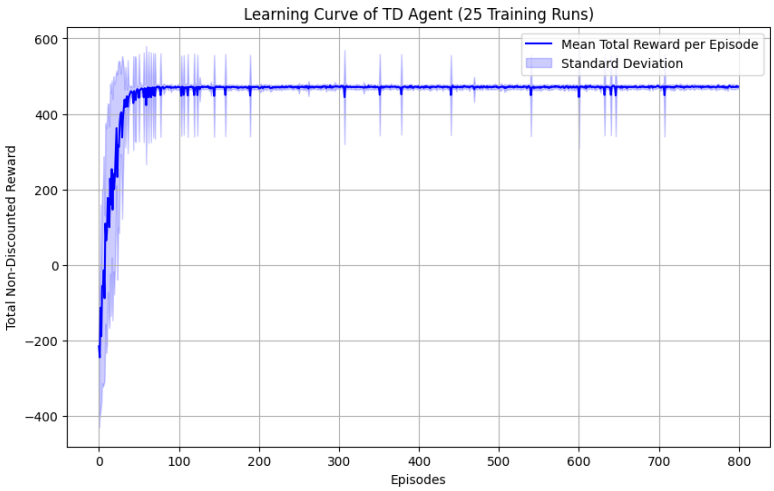

Figure 7: TD Learning Curve

d) **No**, off-policy algorithms (like Q-learning) don't need to be greedy in the limit with infinite exploration(GLIE). GLIE is a condition specifically important for on-policy algorithms, where the policy being improved is also the one being followed to generate experience. GLIE ensures sufficient exploration in on-policy settings, gradually leading to a greedy policy over time. However, off-policy algorithms, such as Q-learning, do not require GLIE because they rely on *behavior* policies that are independent of the *target* policy being evaluated and improved. In off-policy methods, the behavior policy's role is only to ensure exploration across all actions and states. Therefore, the behavior policy can continue exploring, while the target policy (which we aim to improve) converges towards the optimal policy. Thus, as long as the behavior policy covers all actions in all states adequately (satisfying the exploration requirement), off-policy algorithms can learn the optimal action-value function without needing GLIE.