Question 1: Dynamic programming

1.1

Dynamic Programming (DP) is ideal for this finite state and action space maze problem, where each cell represents a distinct state, and actions are finite, with absorbing states present. DP requires complete knowledge of the Markov Decision Processes (MDP), including transition probabilities (T) and the reward function (R), which are available for use here. For this DP problem, it was possible to either use the Policy-Iteration algorithm or the Value Iteration algorithm to solve this maze. After conducting a mean average execution time for these two algorithms, it was shown that they were relatively very similar (0.891 second for Policy Iteration and 0.942 second for Value Iteration). Given this slight edge in speed, Policy Iteration was chosen to derive the optimal policy (π_*) and optimal value function (v_*) for solving the maze environment.

For the policy evaluation threshold (Δ), various thresholds were tested to determine which one accelerated the process while still producing the optimal policy and value function. It was discovered that setting the threshold to 0.01 provided both high precision in determining the optimal policy and quick convergence compared to lower delta values that would have extended the time required to exit the loop. A higher delta would make the convergence quicker, but in different maze conditions, it could lead to less precision.

1.2.

The figure 1 is shown below on the next page

1.3

When γ is close to 0, the agent prioritizes immediate rewards over long-term rewards, favoring actions with short-term benefits. As γ increases, once it reaches a value greater than 0.5, the agent places a higher value on future rewards, leading to exploration of actions with lower immediate gains but higher long-term rewards. In this maze, a higher γ promotes exploration and the discovery of the optimal path, even if it involves actions with lower success probability, denoted as p.

On the other hand, when γ is constant and p is less than 0.25, the agent will have a lower chance of success, as it only has a probability of p to succeed in its chosen action. This leads to more deviations from the intended path and will result in the agent moving away from the optimal policy in the policy graph. When p is equal to 0.25, the environment is balanced since it has an equal chance of success for the chosen action or an alternative one. Due to the randomness of actions taken, the agent will still face difficulties in finding the optimal policy and value function. However, when p has a greater value than 0.25, the agent's actions are more likely to succeed, reducing the likelihood of deviating from its intended path.

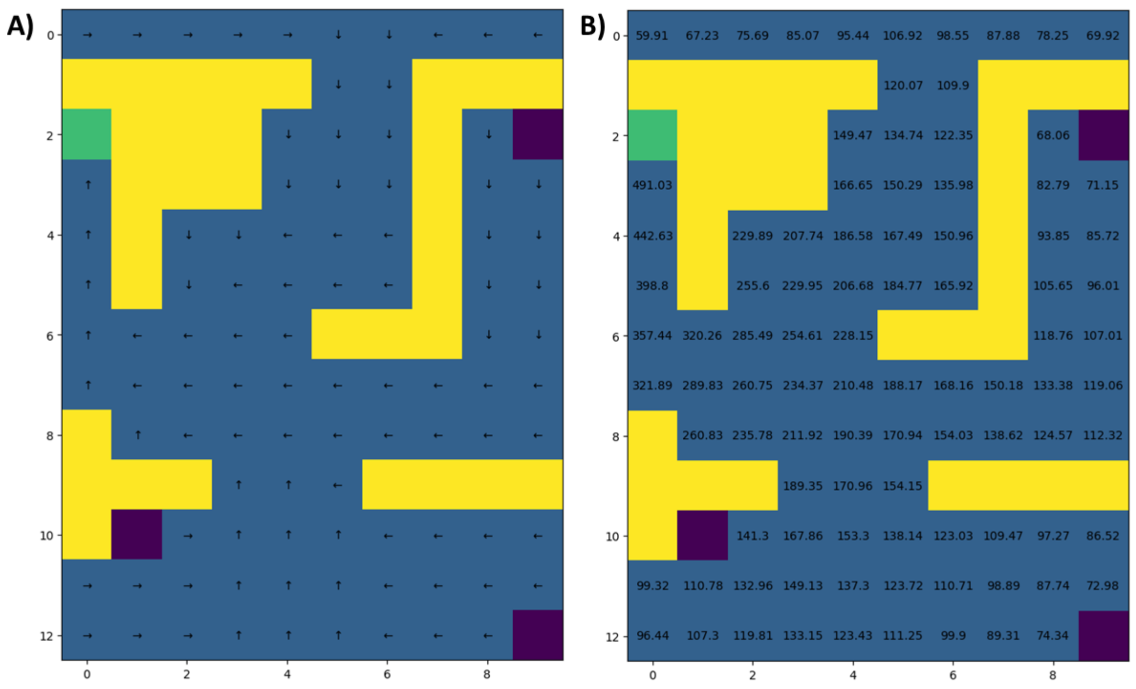


Figure 1: Visualization of the optimal Policy and optimal Value function for the Dynamic Programming Agent. Both graph were generated using the Policy Iteration algorithm with specific parameters: $\gamma = 0.92$, $p = 0.86$ and $\Delta = 0.01$. **A)** Graphical representation of the optimal policy **B)** Graphical representation of the optimal value function

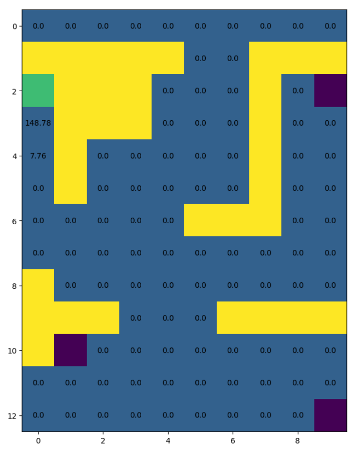
Considering each individual scenarios

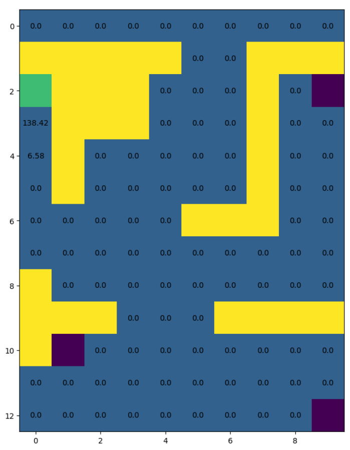
- ($p < 0.25$, $\gamma < 0.5$): The agent faces a low probability of success in its chosen actions and heavily discounts future rewards. As a result, the agent will converge to an optimal policy where each action leads away from the reward state, and the value function will approach nearly zero for most states, with higher values assigned to states adjacent to the reward state. The optimal value function is depicted in Figure 2.a

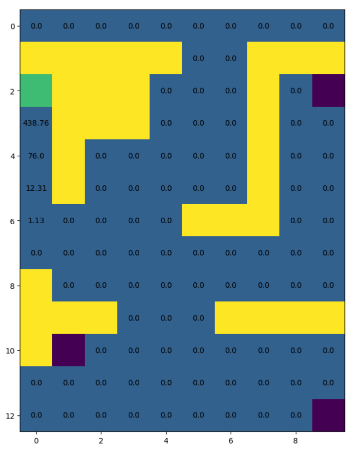
- ($p = 0.25$, $\gamma < 0.5$): The agent faces an equal chance of success and failure in its actions, and the discount factor is lower than 0.5, indicating a lesser emphasis on future rewards. Consequently, the optimal value function will closely resemble the previous scenario, but with slightly lower values adjacent to the reward state due to the stochastic nature of each action. The optimal value function is displayed in Figure 2.b.

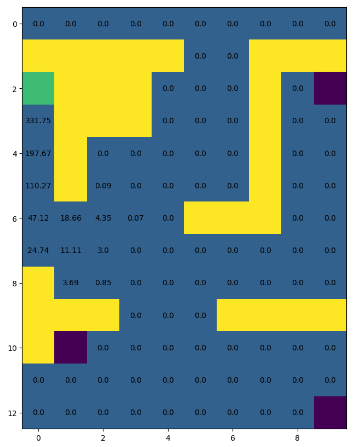
- ($p > 0.25$, $\gamma < 0.5$): The success probability is high, but the discount factor is low. The optimal policy will prioritize paths with higher immediate rewards, even if they involve little risk. This will lead to a value function with higher values only next to the reward state and will be further expanded compared to the next 2 previous scenarios. The optimal value function is shown in Figure 2.c

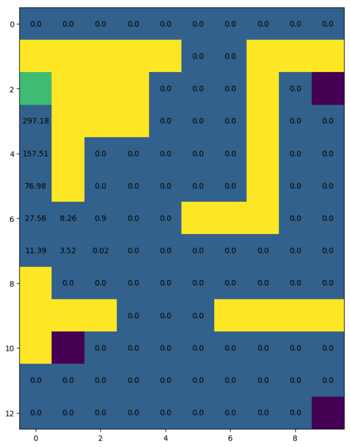
- ($p < 0.25$, $\gamma > 0.5$): The agent encounters a low probability of success in its chosen actions but places a higher value on future rewards, despite the challenges


(a) $\gamma = 0.2$, $p = 0.2$


(b) $\gamma = 0.2$, $p = 0.25$


(c) $\gamma = 0.2$, $p = 0.86$


(d) $\gamma = 0.92$, $p = 0.2$


(e) $\gamma = 0.92$, $p = 0.25$

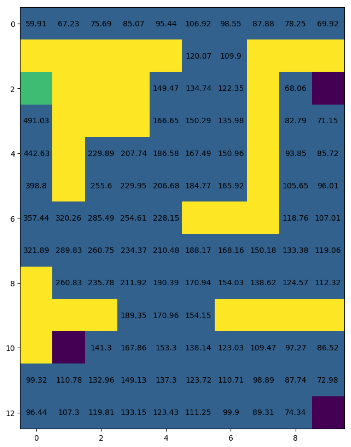

(f) $\gamma = 0.92$, $p = 0.86$

Figure 2: Value Function for various γ and p for the Dynamic Programming Agent

posed by the low success probability. In this case, the optimal policy will still direct the agent away from the reward state. The optimal value function will exhibit high values in proximity to the reward state. The corresponding optimal value function is presented in Figure 2.d.

- (p = 0.25, $\gamma > 0.5$): The agent faces an equal chance of success and failure, yet it places a greater emphasis on future rewards. Consequently, the optimal value function will closely resemble the previous scenario, but with somewhat reduced values, reflecting the inherent randomness in the agent's actions. The optimal value function is depicted in Figure 2.e.

- (p > 0.25, $\gamma > 0.5$): Here both the success probability and the discount factor are high. The optimal policy will prioritize long-term rewards, and the agent will easily navigate within the maze. This results in convergence from all states towards the reward states, leading to very high values in proximity to the reward state. The optimal value function is illustrated in Figure 2.f.

Question 2: Monte-Carlo Reinforcement Learning

2.1

I have chosen to implement the Monte-Carlo algorithm with an on-policy first-visit approach and ϵ-greedy policies to estimate the optimal policy (π_*) and optimal value function (v_*) for solving this maze environment. Utilizing the on-policy first-visit method requires less data storage and is particularly well-suited for episodic tasks with well-defined beginning and end points. It involves online updates, which means regular policy updates after each episode. The choice of an ϵ-greedy policy is motivated by the need to balance exploration and exploitation. Initially, ϵ is set to a high value (starting value of ϵ is set to 0.6), assigning a probability of 0.55 to choosing the best action while allocating a probability of 0.15 to each of the other 3 available actions when updating the policy. This approach allows for exploration of new paths and avoids over-reliance on sub-optimal paths. As training progresses, ϵ gradually decays after each episode, eventually reaching a value close to 0 towards the last episodes. This encourages the agent to follow the learned optimal policy. This balance between exploration and exploitation ensures that Monte-Carlo converges to the optimal policy by implementing exploring starts. The number of episodes was set to 1000 to enable the agent to explore various starting points and accumulate experiences for learning.

2.2

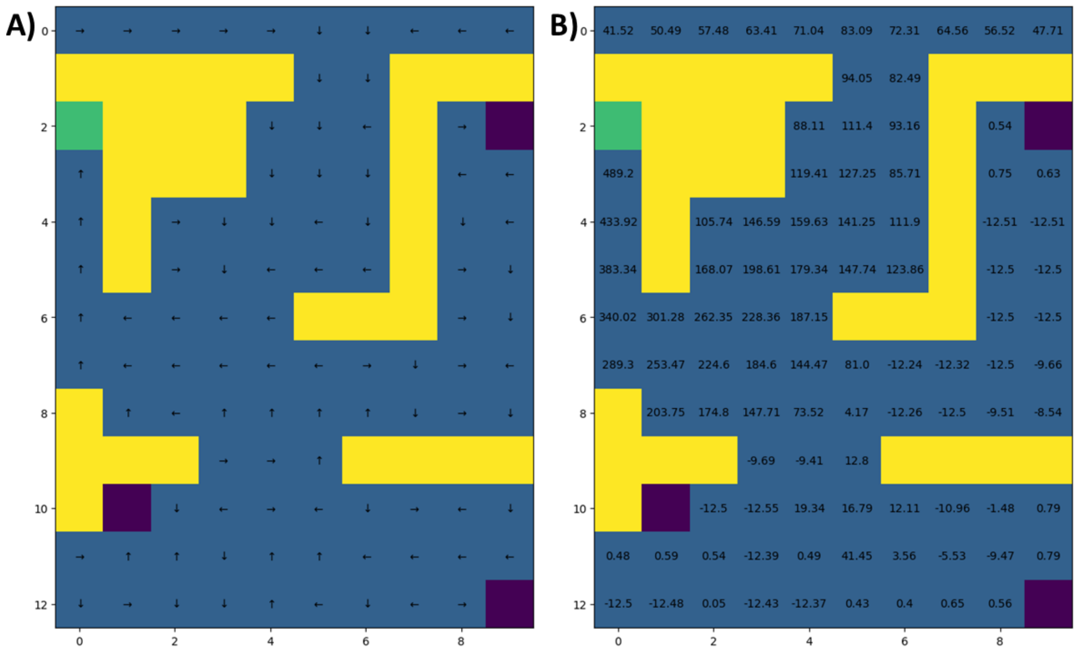


Figure 3: Both graph were generated using the Monte Carlo algorithm with on-policy first visit and ϵ-*greedy* policy with specific parameters: $\gamma = 0.92$, $p = 0.86$, initial $\epsilon = 0.6$ and number of episodes = 1000. **A)** Graphical representation of the estimated optimal policy **B)** Graphical representation of the estimated optimal value function

2.3

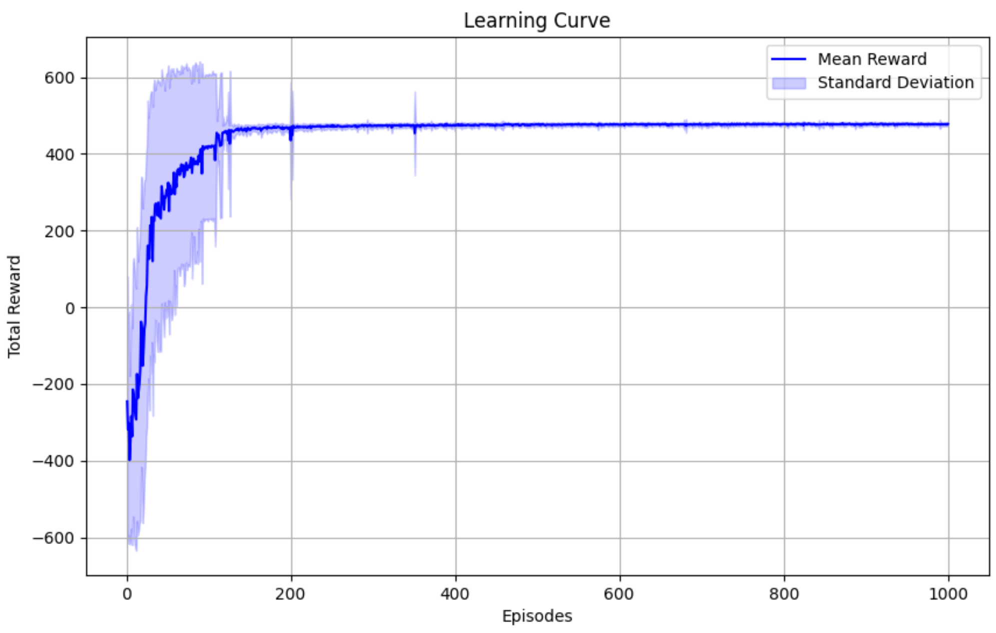


Figure 4: Visualization of the Learning curve for the Monte-Carlo agent. This graph was generated after 25 training runs using the Monte Carlo algorithm with on-policy first visit and ϵ-*greedy* policy.

Question 3: Temporal Difference Reinforcement Learning

3.1

The choice between SARSA and Q-learning was available, and I opted to implement the Q-learning algorithm. Q-learning is an off-policy learning algorithm, meaning it learns the optimal policy from a behavior policy while following a different policy (often epsilon-greedy). This allows it to explore more extensively than SARSA, which is an on-policy method.
After conducting tests with various learning rates (α), as illustrated in Figure 6.b, the learning rate was set to 0.2. This rate facilitates rapid learning and retains the importance of previous episodes. Additionally, an epsilon-greedy strategy was employed with no decay, setting ϵ to 0.2 after extensive testing, as depicted in Figure 6.a. This setting ensures that the agent explores new actions 20% of the time while primarily adhering to its learned policy. The number of episodes was also fixed at 1000 to provide the agent with opportunities to learn from multiple distinct full episodes.

3.2

the figure 5 is shown below on the next page

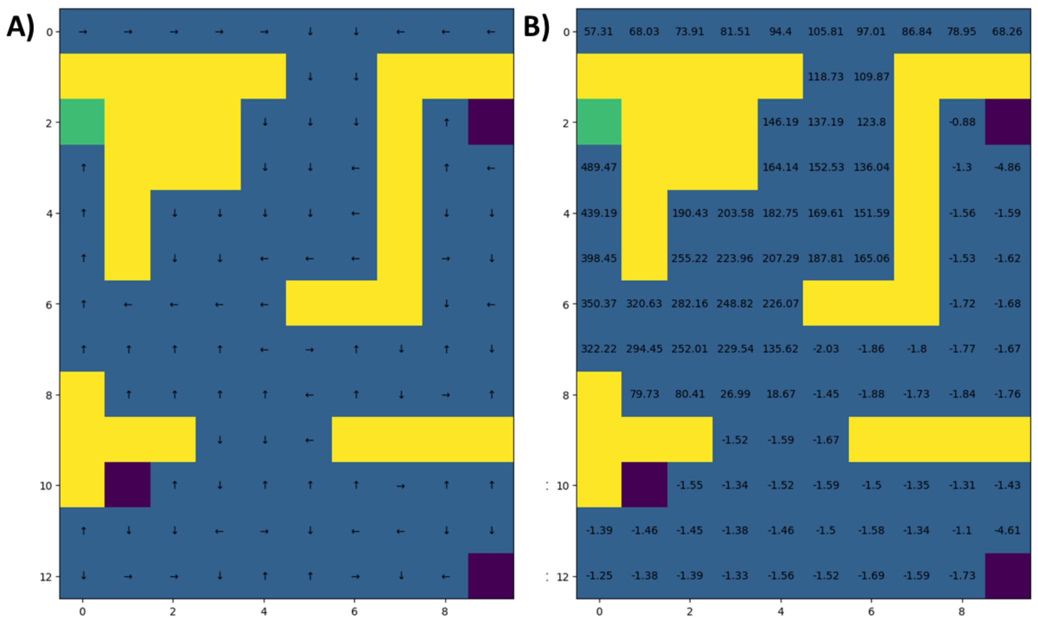
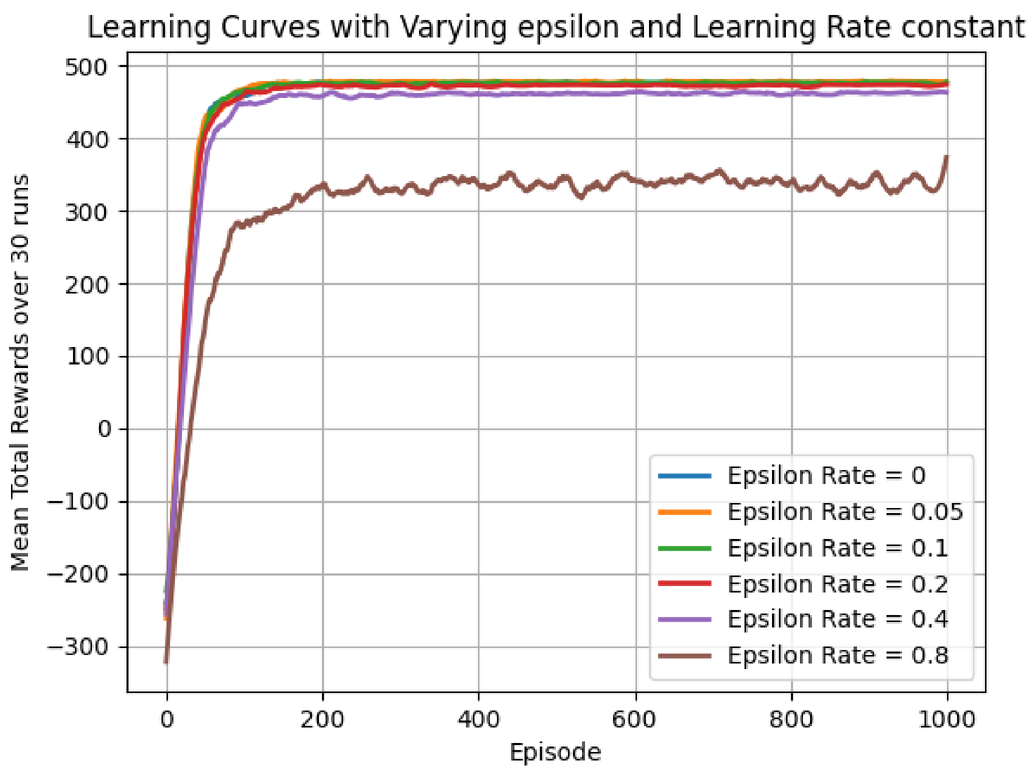


Figure 5: Both graph were generated using the Monte Carlo algorithm with on-policy first visit and ϵ-*greedy* policy with specific parameters: $\gamma = 0.92$, $p = 0.86$, $\epsilon = 0.2$, learning rate $(\alpha) = 0.2$ and number of episodes = 1000. **A)** Graphical representation of the estimated optimal policy **B)** Graphical representation of the estimated optimal value function

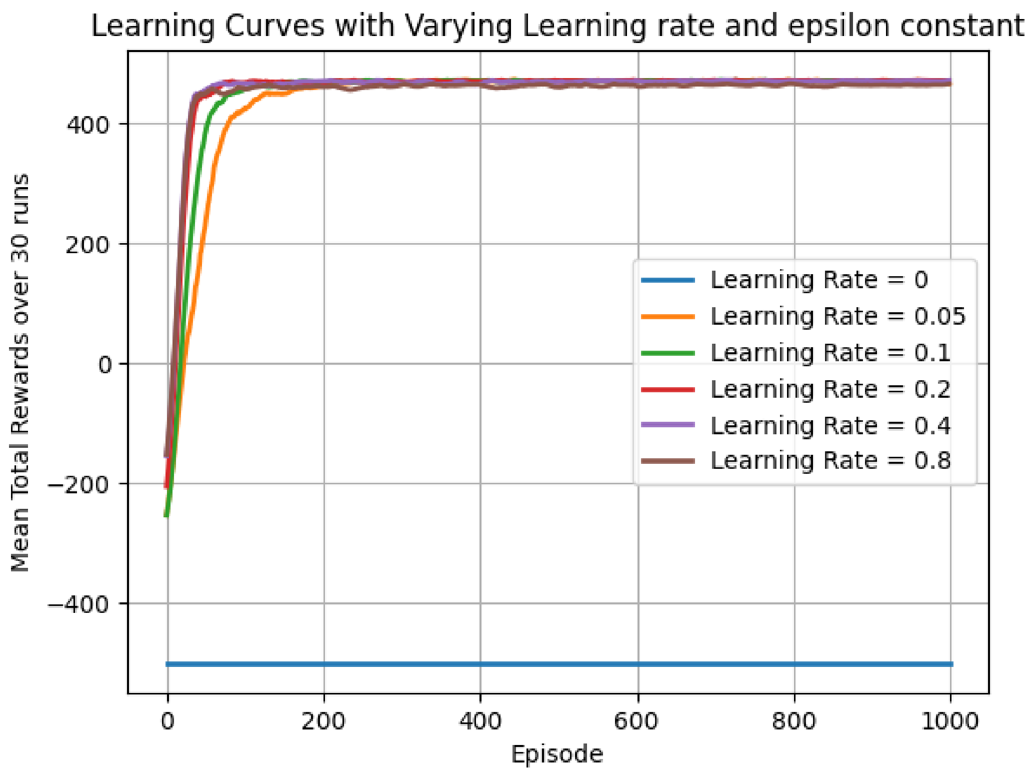
3.3

Starting with a small ϵ, such as 0.05, encourages the agent to exploit its existing knowledge by selecting actions with the highest estimated Q-values. This results in stable learning and optimal policy convergence. However, if ϵ is set to a higher value, like 0.8, the agent is more inclined to take random actions, which promotes exploration of the environment but it will have a hard time to take the optimal action due to high epsilon. In this case, the learning curve struggles to converge to an optimal policy due to excessive exploration. This also leads to an increasing variance in rewards, as depicted in Figure 6.a, as ϵ increases.

Similarly, starting with a small α, like 0.05, implies that the agent relies more on existing knowledge, resulting in slower Q-value updates in a maze. Given the presence of more negative and delayed rewards in the maze, this slower adaptation means it takes longer for the agent to avoid actions that lead to negative rewards and to converge to the optimal policy, as shown in Figure 6.b. Conversely, a higher learning rate, such as 0.2, results in more significant Q-value updates, enabling the agent to learn more quickly to avoid actions leading to negative reward states. However, a very high α," for example, 0.8, can lead to increased variance due to the high weight given to each new episode during Q-value updates.



(a) Mean total reward of various ϵ while keeping α constant



(b) Mean total reward of various α while keeping ϵ constant

Figure 6: Effect of various ϵ and α on learning curves for the TD method