Question 1: Dynamic programming

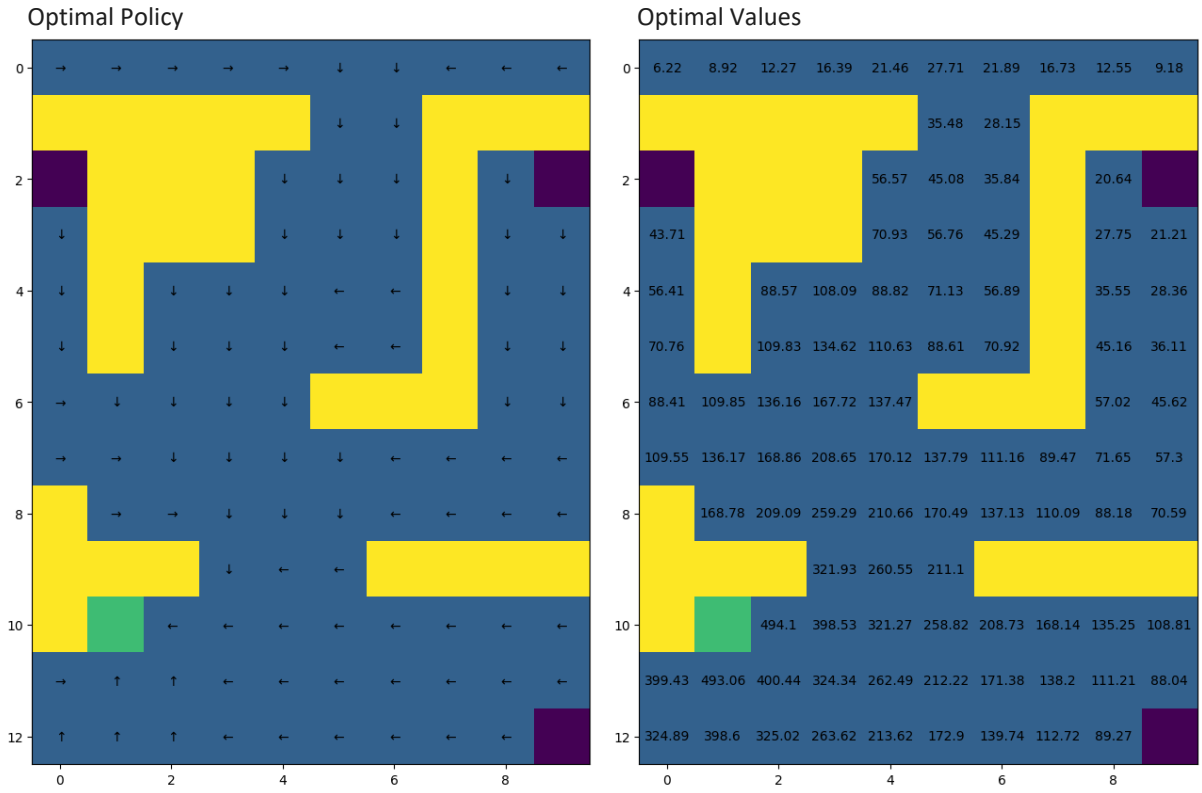
a. State which method you choose to solve the problem. Give and justify any parameters that you set.

Method chosen: value iteration

Parameter chosen and justification:

- Threshold = 0.0001
- The threshold value determines when the value iteration should stop. In particular, when delta (changes in value between iterations) becomes smaller than the set threshold, we stop the updates. Since value function converges to its true values, a small threshold value (0.0001) means the resulting value functions are sufficiently close to the true value of the system. The value of 0.0001 also balances the trade-off between accuracy of state value and computation time. Values smaller than this would require longer training time.

b. Provide the graphical representation of your optimal policy and optimal value, which can be generated with the provided code to visualise policies and value functions. – 2 pts



c. What is the effect on the optimal policy and value function when rewards are scaled by a factor of 2? – 2 pts

The optimal policy will remain unchanged, but the value function will be scaled by a factor of 2. This is because in value iteration method, value functions are set to the maximum of total rewards (rewards from action and discounted value of the next state, subject to transition probabilities).

When the total reward*2, the value function will be scaled by 2 as well. However, this does not affect the relative value of each state, therefore the optimal policy will remain the same.

d. Suppose you now wish to design the reward function such that, for any state s, the value for that state is equal to the probability of eventually reaching the goal state, when starting in state s. Specifically, you are tasked with finding an appropriate choice of:

• discount factor γ

• reward for steps that don't reach a terminal state

• reward for reaching the goal state

• reward for reaching other terminal states

that would result in such a value function. – 3 pts

γ = 1

Reward for steps that don't reach a terminal state = -1

Reward for reaching the goal state = 1

Reward for reaching other terminal states = -1

Setting the value of a state to be the probability of eventually reaching the goal state from that state means that we want the value of the state s to be high if it is highly likely that the agent will reach the goal state from a state s. Since the value of a state s is the expected return from this state, which is the sum of immediate reward and discounted future rewards. The value **γ** reflects how much future rewards are valued in comparison to immediate rewards. Setting **γ = 1** means we value long term outcomes heavily, which means that the overall value function focus heavily on the likelihood of eventually reaching the goal rather than focusing only on the immediate rewards of taking the current step.

Since we want the agent to reach the goal state eventually, we would set the reward of any steps taken or any state reached that does not result in reaching goal as –1. And we set the reward of reaching the goal to be a positive value 1. This combination encourages the agent to continue navigating to the goal state and discourage any unnecessary steps and achieve a high reward.

Mathematically, suppose we start at goal state, then V(s) =1. But V(s) = R + γ*V(s'), where R is the immediate reward of moving, and V(s') is the reward of next state. If the agent stays at terminal, then we have R + γ*V(s') = 0 + 1*1 = 1 = V(s). If the agent navigates away from terminal state, then R + γ*V(s') = -1 + 1*(-1) = -2 < 1. This means that the optimal policy would be to stay at terminal state.

Question 2: Monte-Carlo Reinforcement Learning

a. State which particular MC algorithm you choose to solve the problem. Give and justify the value of any parameters that you set.– 3 pts

MC algorithm chosen: On-policy every-visit Monte Carlo control with epsilon-greedy policy

Parameter chosen and justification:

- epsilon_0 = 1

- decay_factor = 1000

- epsilon_0 is the starting epsilon value and decay_factor is the speed of epsilon decrease. For this MC agent, I have set a decaying epsilon so that at the beginning, the agent always chooses actions randomly. But as we progress through more episodes, the epsilon value decreases based on this formula:

  epsilon = epsilon_0 / (1 + episode_num / decay_factor)

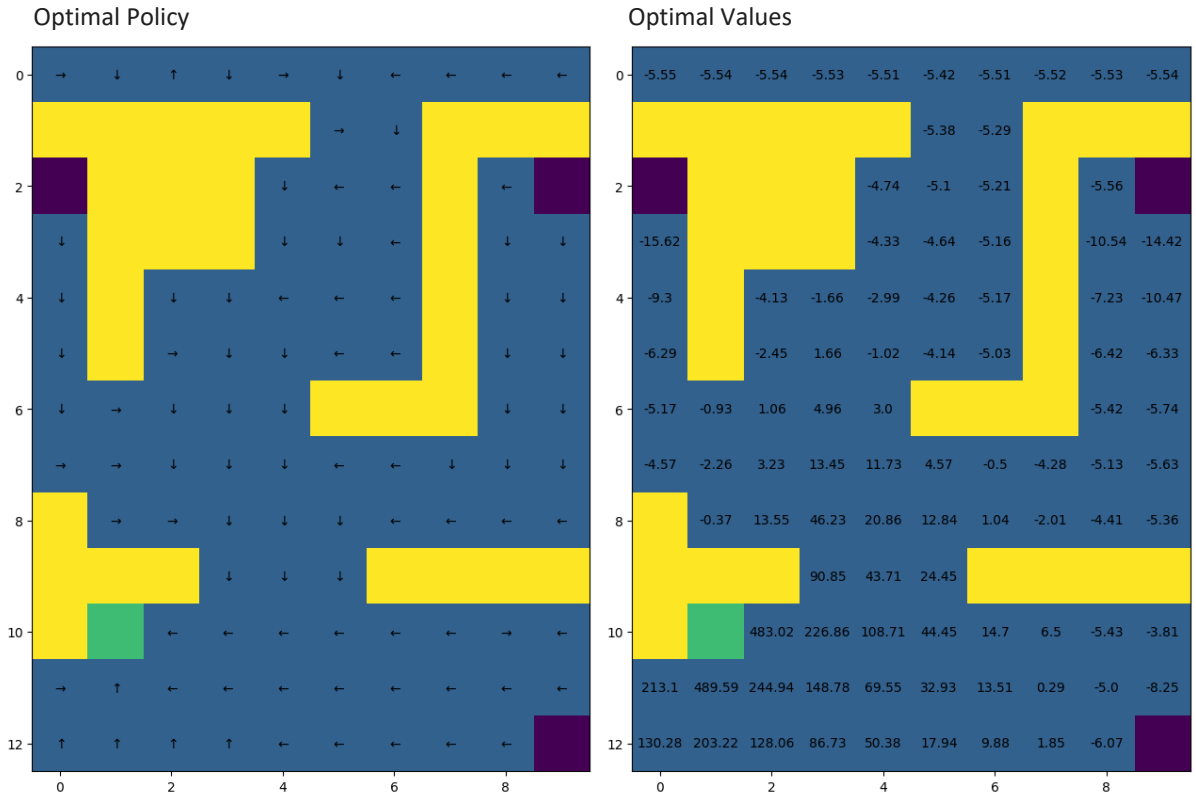
  This set up satisfies GLIE conditions, this ensures the MC algorithm converges to optimal action-value function:

  1. All the state-action pairs are explored infinitely many times:
     The decay factor is set to be a large value (1000), which means the epsilon decreases slowly as we go through more episodes. This allows the agent to explore more state-action pairs during the learning process. (i.e. takes random actions sufficient number of times). Eventually all state-agent pairs will be explored infinitely many times.
  2. The policy converges on a greedy policy:
     For very large episode_num, the epsilon value tend to 0. This means that eventually, the agent will always take greedy policy.

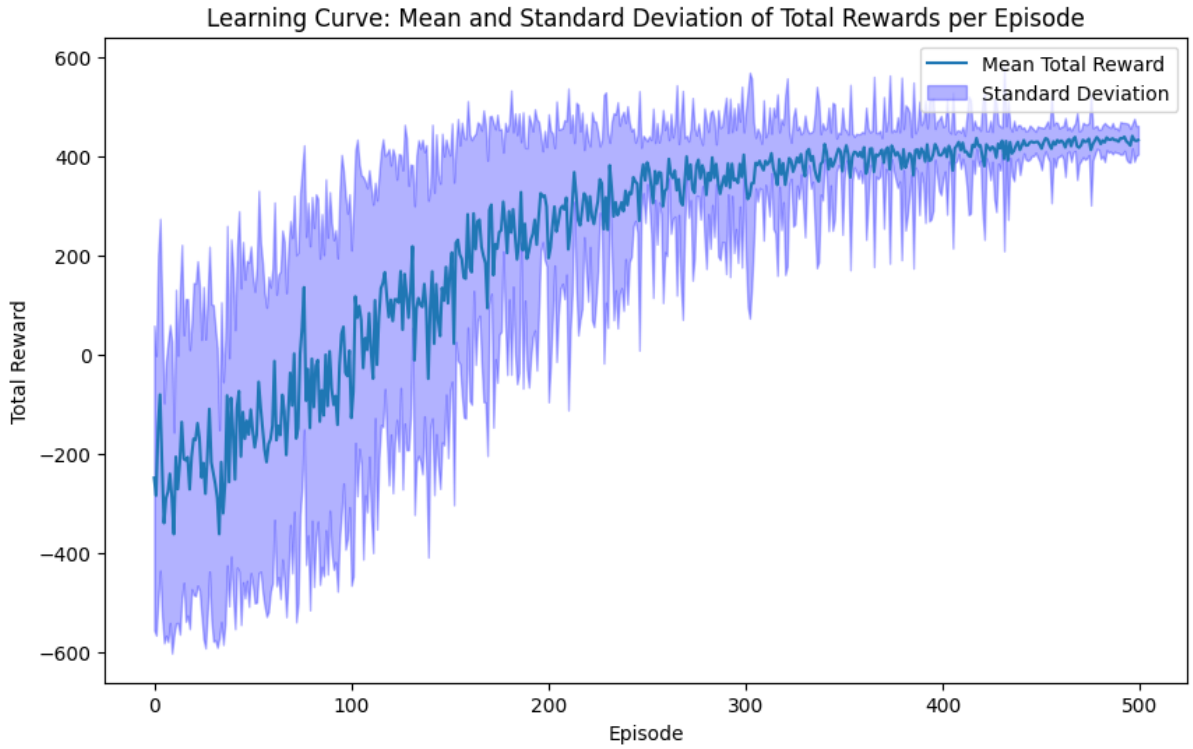
- episodes = 500

- Number of episode is set to be 500 as it both captures convergence pattern and does not extend the running time significantly. A smaller episode number doesn't fully capture the convergence as the total reward is still increasing after the last episode. A large episode led to very long and inefficient running time.

b. Provide the graphical representation of your optimal policy and optimal value function, which can be generated with the provided code to visualise policies and value functions.– 2 pts



c. Plot the learning curve of your agent: the total non-discounted sum of rewards (vertical axis) against the number of episodes (horizontal axis). The figure should show the mean and shaded standard deviation on the same graph across 25 training runs.– 2 pts



d. Consider an implementation which ignores trajectories that don't reach terminal states within 500 steps. In what ways would this affect the MC estimate of the value function compared to an implementation that does not impose a step limit per episode? You do not need to run experiments or provide plots, instead answer and give your reasoning based on your understanding of the lecture material. – 3 pt

The MC estimate of value function will be too high if the specified trajectories are ignored. Those trajectories where the terminal state was not reached within 500 steps will incur lots of negative returns. As value functions are the mean of returns, ignoring these trajectories with low returns means the overall value function will be overestimated. Also, ignoring such trajectories might means that we might be overestimating the likelihood of reaching terminal states from some states, if those states have lots of trajectories longer than 500 steps which are disregarded. This also leads to overestimation of values for these states.

The MC estimate of value function convergence will also be smoother. These disregarded episodes would have lower values. The total rewards for these episodes will be lower in comparison to those reaching terminal states within 500 steps. Ignoring these episodes means that there will be less 'large negative spikes' in total reward values. The MC estimate of the value function will converge smoother.

Question 3: Temporal Difference Reinforcement Learning– 15 pts

a. State which particular TD learning algorithm you choose to solve the problem. Give and justify the value of any parameters that you set.– 3 pts

TD learning algorithm chosen: Q-learning

Parameter chosen and justification:

- epsilon = 0.5

- epsilon is set to be constant 0.5 so that throughout the training process, the agent will choose actions randomly half of the time and act greedily the other half of the time. This choice ensures sufficient exploration of the space, as well as exploitation of optimal actions.

- episodes = 500

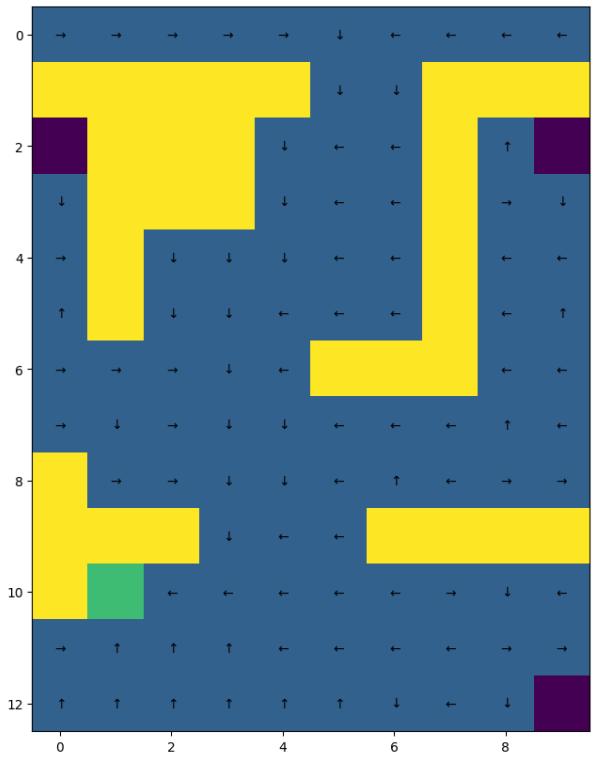
- Number of episode is set to be 500 as it both captures convergence and does not extend the running time significantly. A smaller episode number doesn't fully capture the convergence as the total reward is still increasing after the last episode. A large episode led to very long and inefficient running time.

- alpha = 0.1

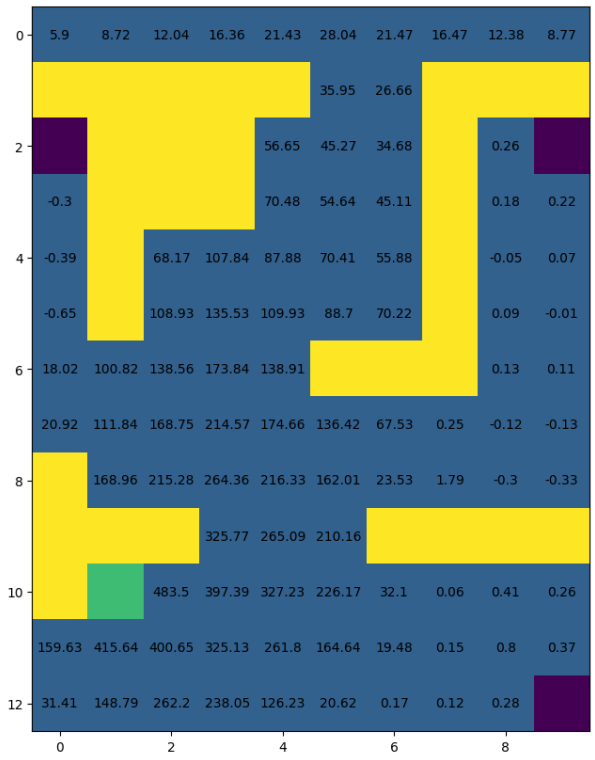
- alpha is set to be the learning rate. It controls the rate at which Q value is updated with new rewards and value of next state. With a relatively lower learning rate (0.1), the Q value is updated only by small amount each time. This gives a relatively stable movement towards convergence of Q value, meaning that it is less sensitive to fluctuations in individual episodes.

b. Provide the graphical representation of your optimal policy and optimal value function, which can be generated with the provided code to visualise policies and value functions.– 2 pts

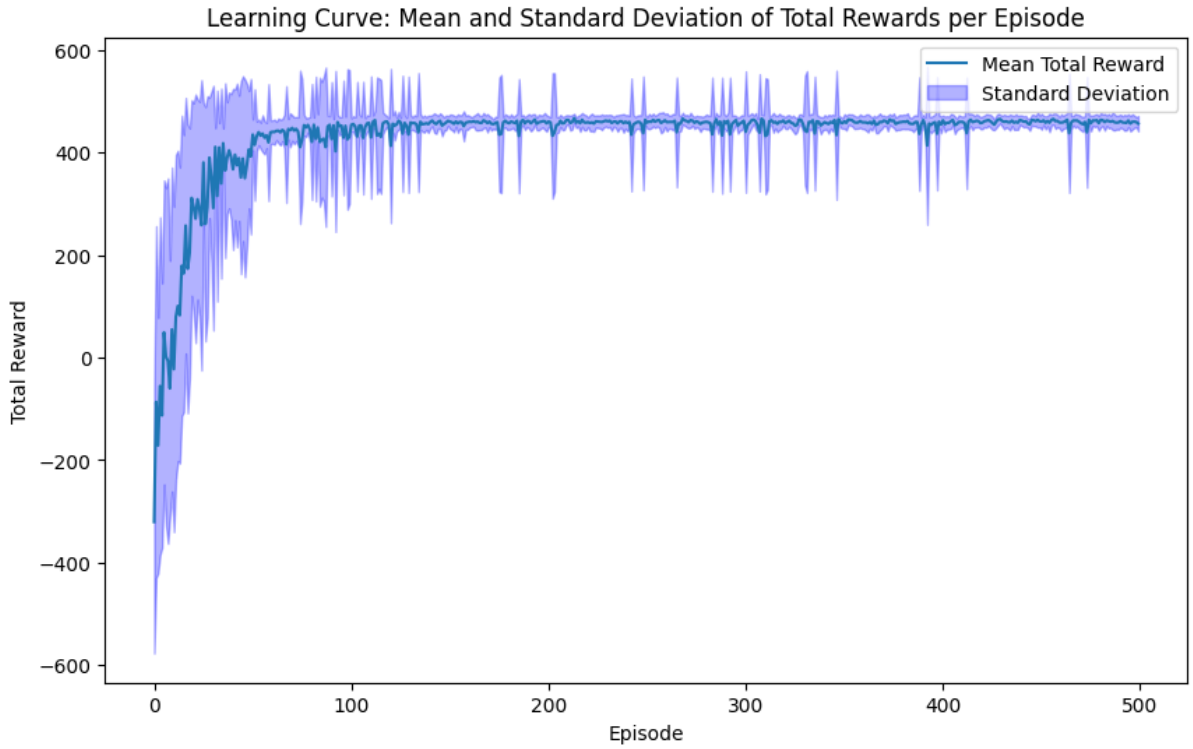
Optimal Policy



Optimal Values



c. Plot the learning curve of your agent: the total non-discounted sum of rewards (vertical axis) against the number of episodes (horizontal axis). The figure should show the mean and shaded standard deviation on the same graph across 25 training runs.– 2 pts



d. Do off-policy algorithms need to be greedy in the limit with infinite exploration (GLIE), in order to guarantee that the optimal action-value function is learned? You do not need to run experiments or provide plots, instead answer and give your reasoning based on your understanding of the lecture material. – 3 pts

No. Off-policy algorithms such as Q-learning separates out target policy π and behavioural policy π'. The behavioural policy is set to explore the environment but it's not necessarily greedy in the long term (as we accumulate more episodes). Since target policy π is always greedy with respect to Q(s,a), we are constantly moving Q value toward the optimal Q* using the best possible Q value in the next state. Therefore, Q eventually converge to Q* even if the second condition of GLIE is not met.