Imperial College London

COMP70028

Reinforcement Learning

October 2023

Imperial College London

Department of Computing

—Coursework 1—

Name:
Kyoya Higashino

Course:

MSc Artificial Intelligence

1 Problem Definition

Description of the Maze environment

The focus of this Coursework is the resolution of a Maze environment, illustrated in Figure 1 and model as a Markov Decision Process (MDP). In this illustration, the black squares symbolise obstacles, and the dark-grey squares absorbing states, that correspond to specific rewards. Absorbing states are terminal states, there is no transition from an absorbing state to any other state.

This environment shares a lot of similarities with the Grid World environment studied in Tutorial 2, you are allowed to reuse any code you might have developed for this tutorial. However, unlike the Tutorial, this Coursework aims to help you implement RL techniques that do not require full knowledge of the dynamic of the environment, namely Monte-Carlo and Temporal-Difference learning. In the following, we will thus assume we don't have access to the transition matrix T and reward function R of the environment, and try to find the optimal policy π by sampling episodes in it.

Consider an agent moving inside this maze, trying to get the best possible reward. To do so, it can choose at any time-step between four actions:

- $a_0 =$ *going north* of its current state
- $a_1 =$ *going east* of its current state
- $a_2 =$ *going south* of its current state
- $a_3 =$ *going west* of its current state

The chosen action has a probability p to succeed and lead to the expected direction. If it fails, it has equal probability to lead to any other direction. For example, if the agent chooses the action $a_0 =$ *going north*, it is going to succeed and go north with probability p, and it has probability $\frac{1-p}{3}$ to go east, probability $\frac{1-p}{3}$ to go south and probability $\frac{1-p}{3}$ to go west. p is given as a hyperparameter, defining the environment.

If the outcome of an action leads the agent to a wall or an obstacle, its effect is to keep the agent where it is. For example, if the agent chooses the action $a_1 =$ *going east* and there is a wall east but no wall in the other directions, it is going to stay in place in his current state with probability p and to go north, south or west with the same probability $\frac{1-p}{3}$.

Any action performs by the agent in the environment gives it a reward of -1.

The episode ends when the agent reach one of the absorbing state $(R_i)_{0 \leq i \leq 3}$ or if it performs more than 500 steps in the environment.

Finally, the starting state is chosen randomly at the beginning of each episode from the possible start states with equal probability $1/N_{start-states}$. The possible start states are indicated on the Maze illustration as "S".

This environment is personalised by your College ID (CID) number, specifically the last 2 digits (which we refer to as y and z), as follow:

- The probability p is set as $p = 0.8 + 0.02 \times (9 - y)$.
- The discount factor γ is set as $\gamma = 0.8 + 0.02 \times y$.
- The reward state is R_i with $i = z \ mod \ 4$ (meaning that if $z = 0$ or $z = 4$ or $z = 8$, the reward state would be R_0, if $z = 1$ or $z = 5$ or $z = 9$, it would be R_1, etc).

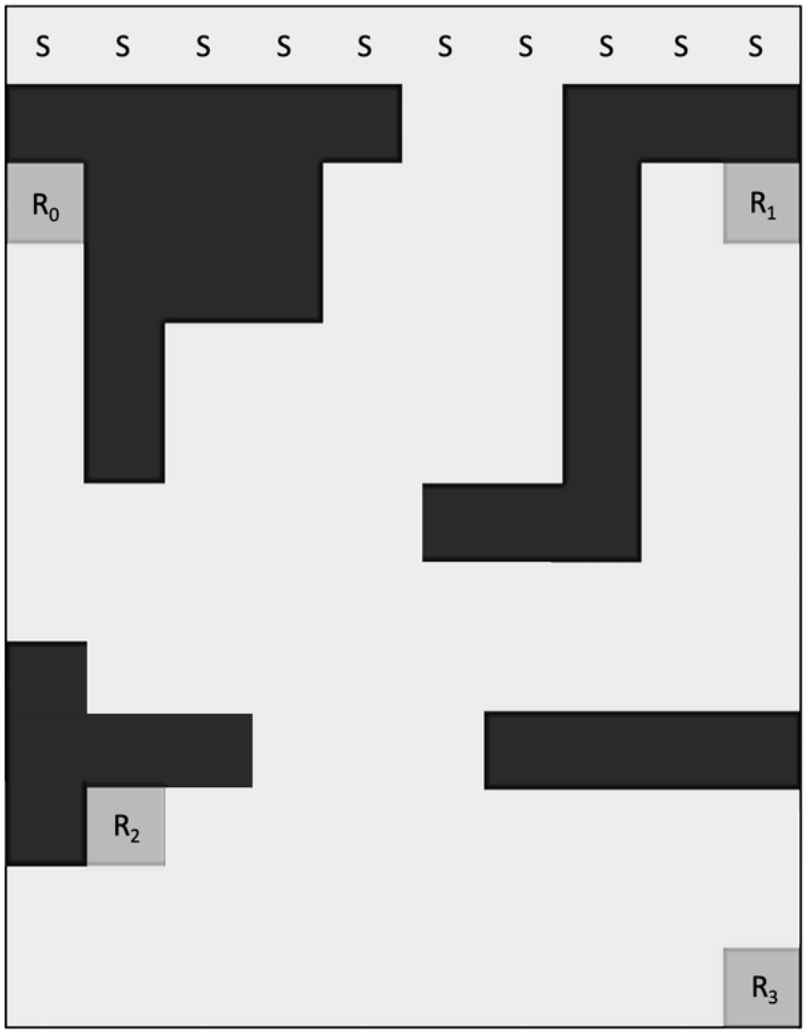

Figure 1: Illustration of the Maze environment, the focus of this Coursework. The absorbing states are indicated as "R_i", they correspond to absorbing rewards. The "S" indicate the potential start states.

- The reward state R_i gives a reward $r = 500$ and all the other absorbing states gives a reward $r = -50$. As stated before, any action performed by the agent in the environment also gives it a reward of -1.

2 Dynamic Programming

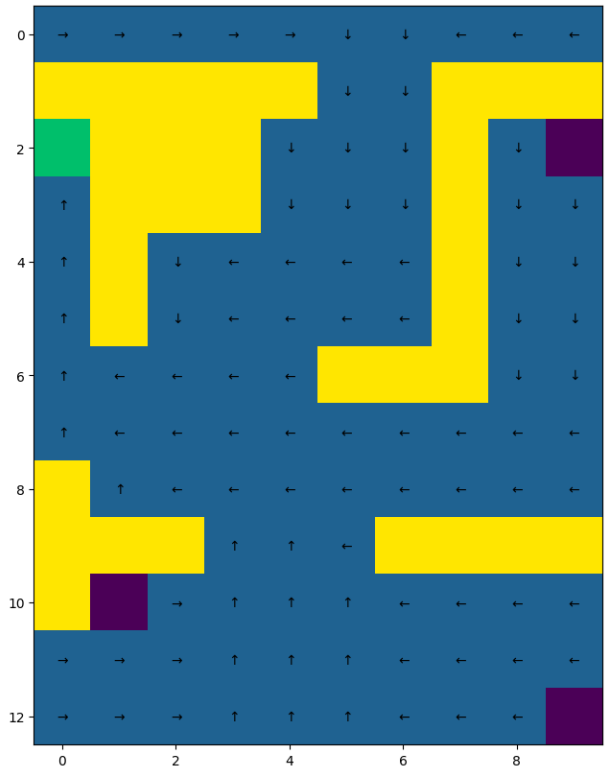

Figure 1: DP optimal policy

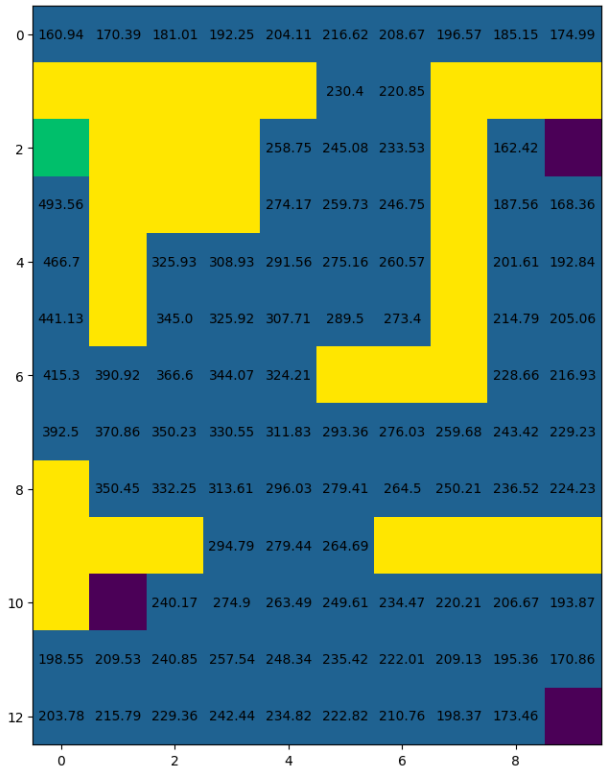

Figure 2: DP optimal value function

2.1 Method Selection and Justification

For dynamic programming, value iteration was selected over policy iteration due to several reasons: increased simplicity and computational efficiency, faster convergence, the maze's small state size, and reduced memory requirements.

Value iteration combines policy evaluation and improvement into a single step, making it simpler and easier to implement than policy iteration. Unlike policy iteration, value iteration updates its values after every iteration, thereby not requiring a convergence condition like in policy evaluation before the policy can be updated. This constant update of state values allows value iteration to reach its optimal value function faster than policy iteration which must complete more iterations as it alternates between policy evaluation and improvement.

The maze's relatively small size (10 x 13) also makes it more suitable for value iteration, able to reach convergence faster as there are fewer states to evaluate. While also true for policy iteration, multiple sweeps of the environment may be required to obtain a stable policy. Conversely, value iteration requires no policy, updating state values based on only

the immediate reward and value of the subsequent state without explicitly calculating or storing any policies. This also reduces value iteration's memory requirements.

2.2 Parameters and Assumptions

The customisable parameters only include the Delta (δ) threshold for establishing convergence. For the optimal policy and optimal value function graphically displayed in Figures 1 and 2, $\delta = 0.001$ was selected, ensuring an extremely close approximation of the true optimal value function. However, its worthy noting that tests with $\delta = 1$ resulted in only a maximum change of 0.5% in certain states, showing that much higher convergence thresholds can also be used in attaining the optimal value function. In fact, δ values up to 35 will still give the optimal policy, albeit not converged to near the optimal value function.

2.3 Effect of γ and p

The parameter p is the probability of success that the agent moves in its intended direction. If unsuccessful, the agent moves in one of the remaining three directions with equal probability. If $p = 0.25$, the agent has no control of its movement as each direction is equally likely regardless of the intended action. As such, no policy can be determined, and all state values are very low. If $p > 0.25$, the agent has some control of its movement, and the optimal policy can be derived. Higher p values also result in higher optimal value functions as it is more likely to traverse the maze according to the optimal policy, reaching the goal faster and therefore increasing the value of every state.

If $p < 0.25$, the agent still has some control of its movement, but results in a completely reversed optimal policy. By having a relatively lower probability of moving in the intended direction, the agent learns to not choose the 'best' (correct) direction. While choosing any of the remaining three actions would result in an equal chance of moving in the 'best' direction, the agent counterintuitively learns to choose the opposite or 'worst' direction. This is because choosing the 'worst' direction also minimises the likelihood of moving in the 'worst' direction while maximising the probability of moving in the 'best' direction.

The parameter γ influences the importance of future rewards, with higher γ values considering future rewards more heavily and encouraging exploration. Hence, higher γ values result in higher optimal value functions as the future final reward is discounted less heavily. At low γ values, future rewards are discounted so severely that the optimal policy is not found for states far away from the final reward. As γ values continue to decrease, states closer to the final reward will also be unable to find the optimal policy as consideration of the future final reward becomes negligible.

3 Monte-Carlo Reinforcement Learning

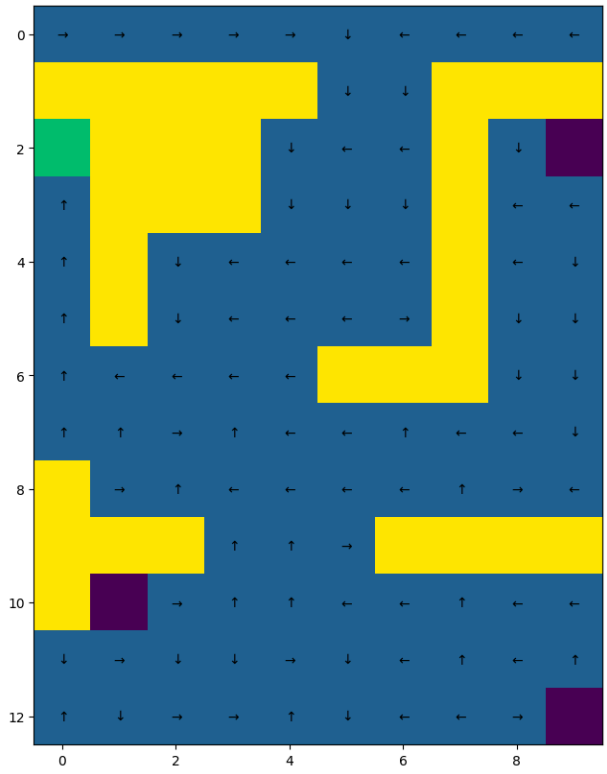

Figure 3: MC optimal policy

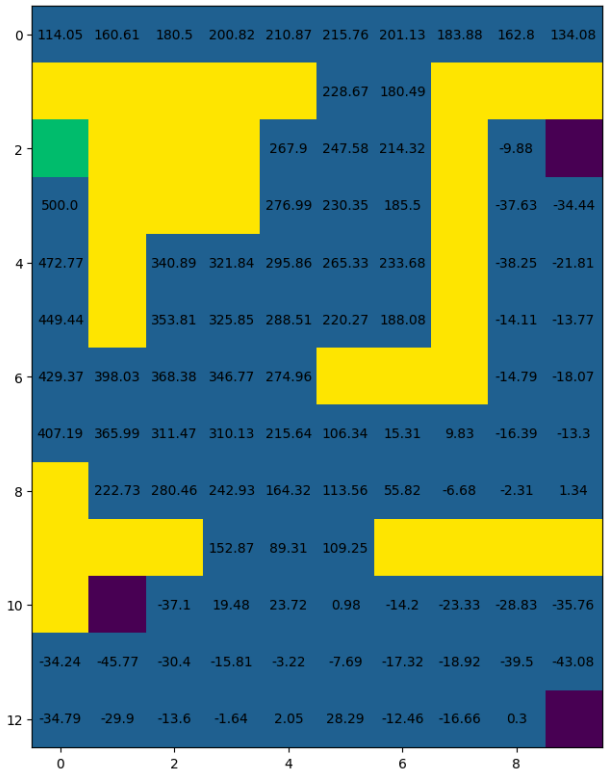

Figure 4: MC optimal value function

3.1 Method Selection and Justification

The employed Monte Carlo method combined several different aspects: on-policy, ϵ-greedy, first-visit, and iterative learning.

The simplicity of the on-policy, ϵ-greedy strategy makes it easy to implement, and as the maze size is relatively small, the complexity of off-policy methods is unnecessary. Off-policy methods also involve maintaining multiple behaviour and target policies, being unnecessarily computationally expensive. The ϵ-greedy policy also allows for direct exploration control, allowing for tuning that can optimise the balance between exploring new states and exploiting the current best policy.

First-visit and iterative learning principles were also incorporated for their benefits in sample efficiency, reduced memory requirements, real-time learning, and simplicity. As the sample space is quite small, many states are often re-visited, making the first-visit method an efficient sampling technique that requires less memory than the every-visit method. As for iterative learning, it was selected for its ability to learn in real-time, quickly adapting to find the optimal policy after every iteration. First-visit and iterative

learning are also simpler to implement and manage, not needing to store large amounts of data or perform batch updates that are unnecessary in a small maze environment.

3.2 Parameters and Assumptions

Customisable parameters include the ϵ value for the ϵ-greedy policy and the total number of episodes. For ϵ, a linear decay was used such that $\epsilon = 1 - \frac{k}{n}$, where k is the current episode iteration and n is the number of episodes. A linear decay was chosen so that the agent takes more exploratory actions at the start of the training run and then more exploitative measures at the end when it is more certain of the best possible action.

The number of episodes was chosen to be 4000 as tests showed that this was the minimum requirement to consistently produce the optimal policy. The optimal policy is assumed to be achieved as long as the cells near the shortest path to the goal display optimal directions. This is because the agent is unlikely to move beyond the shortest path once it is discovered, making a deterministic policy in these faraway areas unimportant.

3.3 Learning Curve Plot

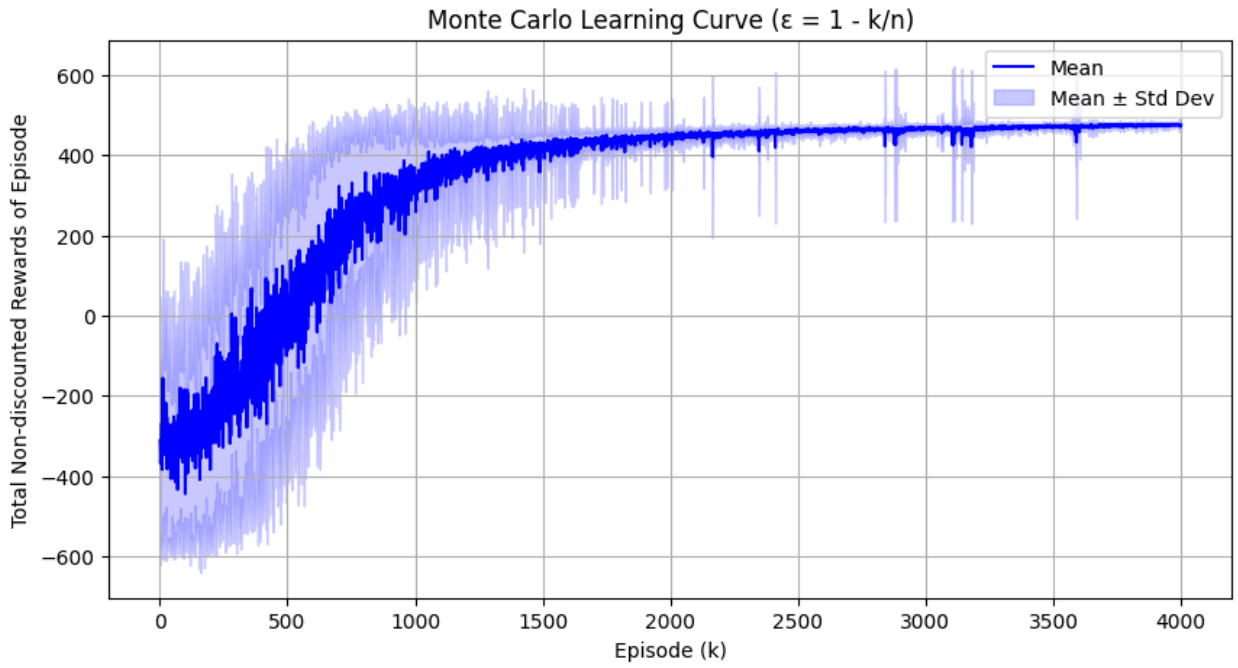

Figure 5: Monte Carlo learning curve over 25 training runs with mean and standard deviation $\left(\epsilon = 1 - \frac{k}{n}\right)$

While convergence takes longer with a decaying ϵ, the resultant deterministic policy is closer to the true optimal policy. As seen in Figure 5, convergence occurs at approximately 2500 episodes and the standard deviation decreases as the number of episodes increases. This is shown as a thinning of the plus/minus standard deviation band (light blue).

4 Temporal Difference Reinforcement Learning

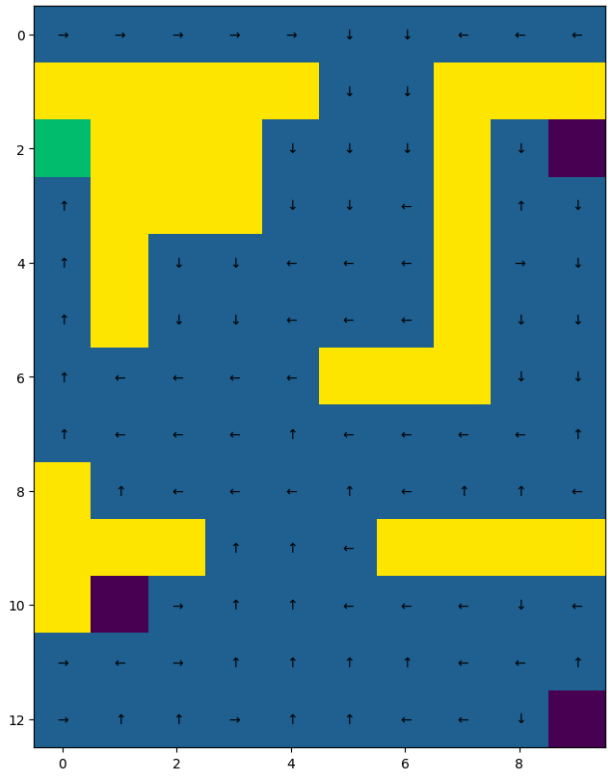

Figure 6: TD optimal policy

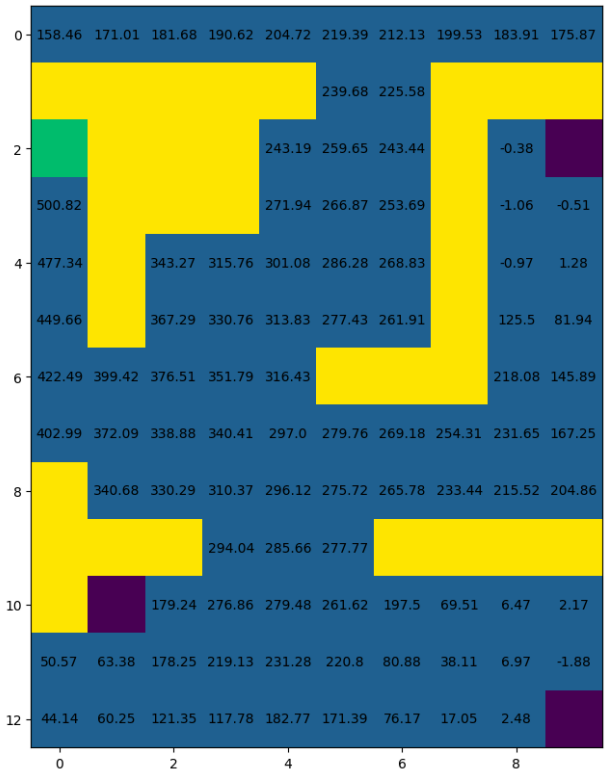

Figure 7: TD optimal value function

4.1 Method Selection and Justification

For the Temporal Difference method, an off-policy technique was employed: Q-learning. This method brings multiple advantages including ease of implementation, exploratory tendencies, and faster convergence in comparison to other TD techniques such as SARSA.

Q-learning's simplicity involving its update rule and consistent greedy policy makes implementation easy. By updating Q-values based on the maximum Q-value of the next state-action pair and the current reward, Q-learning is simple to understand. Conversely, SARSA involves using Q-values of the current and next states based on a current policy, not being as intuitive as considering the next best state-action pair (greedy).

As an off-policy method, Q-learning has better and more flexible exploratory properties. In SARSA, controlling exploration is difficult as it relies on the policy it is currently learning, possibly acting more conservatively in its movement to evaluate the current best state-action pairs. Conversely, the Q-learning method used here also incorporates ϵ-greedy, where ϵ can be tuned to easily control the exploratory tendencies of the agent while still evaluating the best state-action choices (greedy target policy).

Q-learning's exploratory nature also allows it to find the optimal policy faster, leading to faster convergence. Furthermore, it also performs well in more deterministic environments like this maze where the success of selecting an action is high. As it explores the maze, it can more confidently select actions with higher Q-values. In comparison, SARSA's more conservative following of its policy may result in pursuing sub-optimal policies that it initially chose, increasing time to convergence.

4.2 Parameters and Assumptions

Like Monte Carlo, the parameters ϵ (for ϵ-greedy) and n (total number of episodes) are set, including a new parameter α (the learning rate). Again, ϵ is linearly decayed, but n is reduced to 500 episodes as tests showed that the TD agent can find the optimal policy much faster. Similarly, the optimal policy is assumed to be obtained once cells close to the optimal path are determined. The learning rate α has been given a value of 0.5 as tests have shown it to be a good balance between finding the correct optimal policy and rapid convergence. To see the effect of varying α values, see Figure 9 below.

4.3 Effect of ϵ and α on Learning Curve

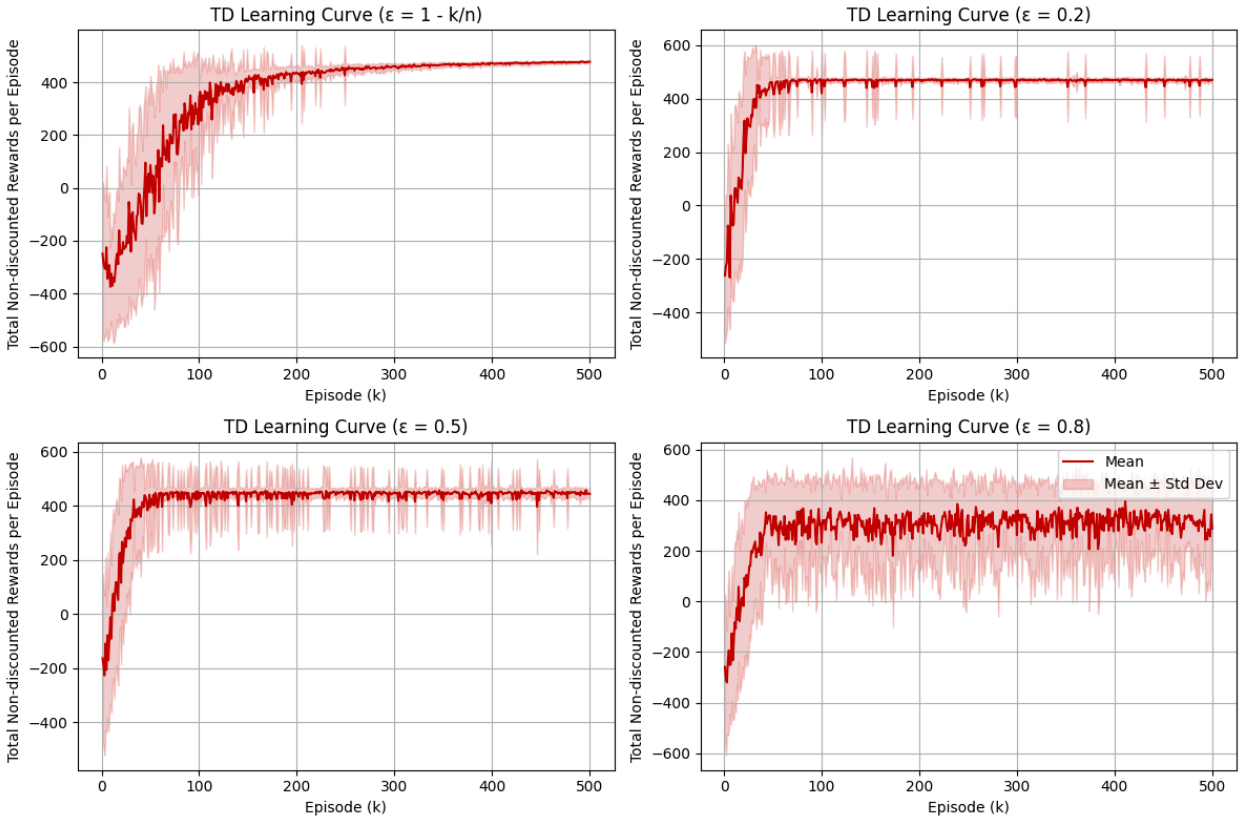

Figure 8: Effect of varying ϵ on TD learning curve (α= 0.5)

In Figure 8, the effect of ϵ is shown to have varying impacts. As stated in Monte Carlo, a linearly decaying ϵ takes longer to converge, but has advantages such as greatly reduced noise and smaller standard deviation. When ϵ is constant, its control over exploratory and exploitative tendencies is seen more clearly. At low ϵ values, the agent is more exploitative, demonstrated by a slightly quicker convergence when $\epsilon = 0.2$. However, this leads to pursuing sub-optimal strategies and errors in the resultant policy due to insufficient exploration. At high ϵ values, the agent is more exploratory, demonstrated by the high variance and thick standard deviation band when $\epsilon = 0.8$. While this ensures sufficient exploration to find the optimal path, exploratory actions continue to be taken in later episodes where the optimal path has already been found, resulting in instability.

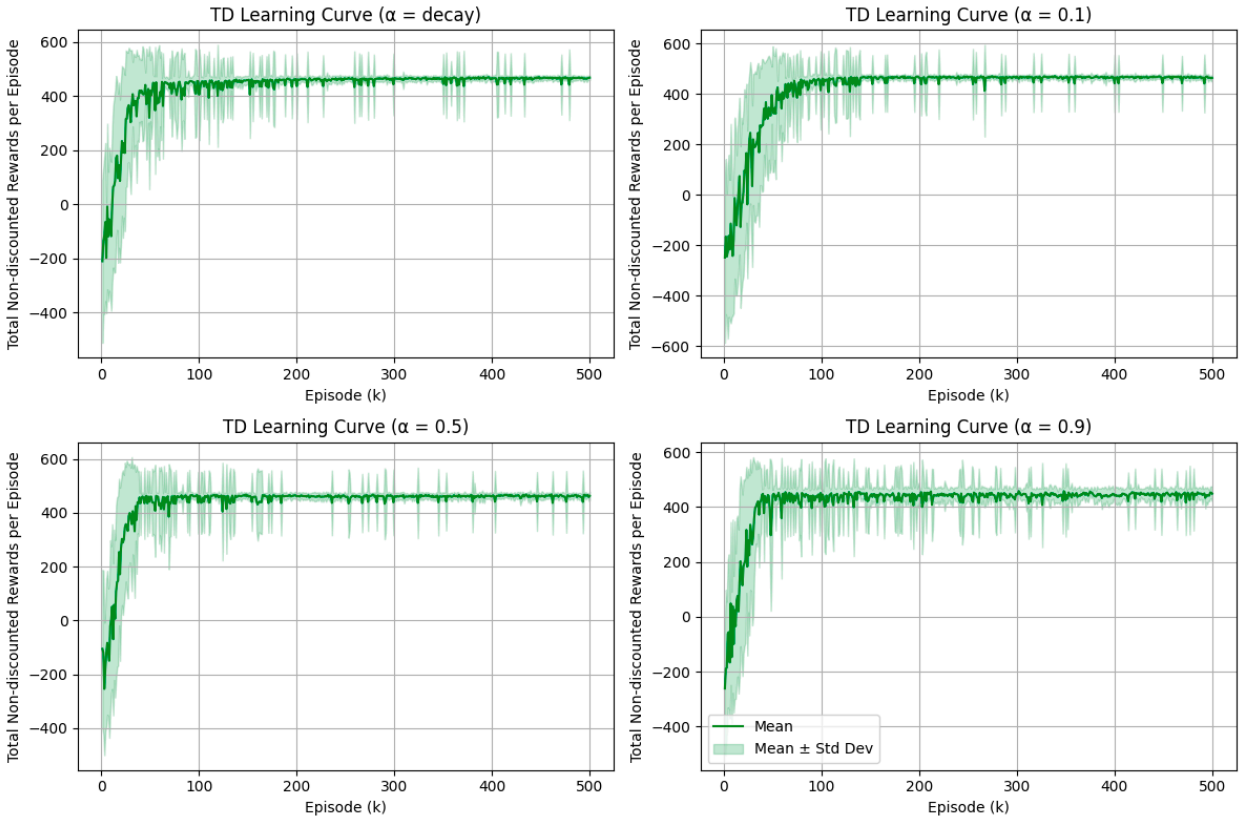

Figure 9: Effect of varying α on TD learning curve ($\epsilon = 0.3$)

Like ϵ, α can also benefit from a linear decay in theory. To emphasise learning from initial exploration, the α can be set high initially and then decayed over time to exploit the learned knowledge from previous episodes. However, this decay has little effect on this TD learning curve, likely due to the reduced size and simplicity of the maze not requiring very sophisticated parameter settings. When α is low, learning is slower as the agent takes more conservative steps toward convergence (i.e. $\alpha = 0.1$). When α is high, convergence is faster but less stable, being more sensitive to noise. While all values of α result in convergence, the reduced stability in higher α learning curves can be seen as a thicker standard deviation band, particularly when $\alpha = 0.9$.