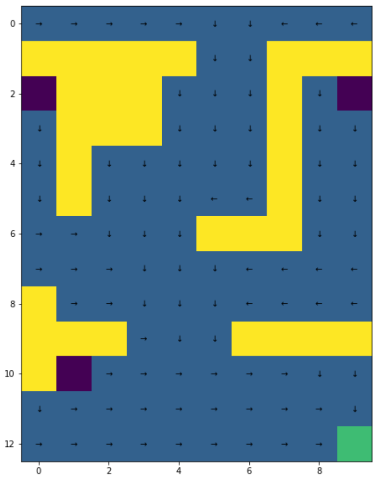
1 Question 1: Dynamic Programming

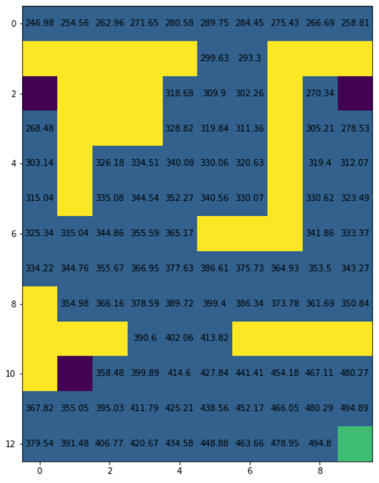
1.1 Choice of Method

I chose to use deterministic value iteration over using policy iteration. Policy iteration evaluates a policy and then uses the values to improve the policy, repeating until the optimal policy is reached. This results in a sub optimal policy to be fully evaluated at each iteration prior to the optimal policy, and a lot of wasted computational effort. Value iteration reduces the evaluation stage down to a single sweep of all the states, giving the maximum expected reward. The time it takes to converge is therefore significantly improved.

1.2 Plots of Optimal Policy and Value Function

In Figure 1, as expected the policy takes us toward the specified target in the bottom right of the map, with the difference in values reflecting this.


(a) DP Policy



(b) DP Value

Figure 1: DP Policy and Value Plots

1.3 Exploring Parameters

The value I have set for γ is 0.98. If p = 0.25 then if the agent 'chooses' an action, it has an equal probability of going in any direction according to how the Transition matrix is updated in the 'Maze' class. Q keeps the same value when updated according to the Bellman equation, V takes the maximum Q which is the first element (go north) by default. The policy reflects this and chooses north each time, and there is no policy improvement. For convergence to optimal policy, we need a value of p $>$ 0.25. If we have p $<$ 0.25 we get the effect that the policy improves but the transition matrix means you are less likely to take the chosen action than the other three. This means we converge on the least optimal path, and the policy reflects this by showing all arrows pointing away from the target. Changing γ has the effect of increasing the time needed to converge, and reflects the fact that for each iteration the Bellman equation takes a bigger slice of the prior state value. This means that for each state there is a bigger change in value and convergence takes longer. It also means that once iterations have completed there is a higher mean value function.

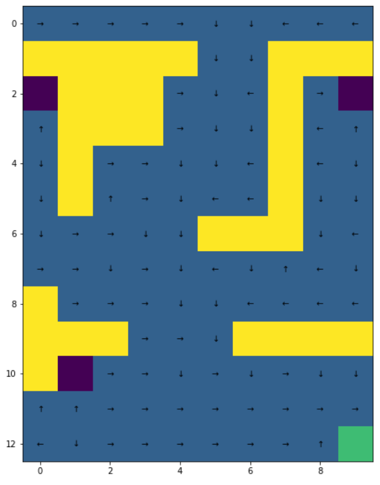
2 Question 2: Monte-Carlo Reinforcement Learning

2.1 Method Justification

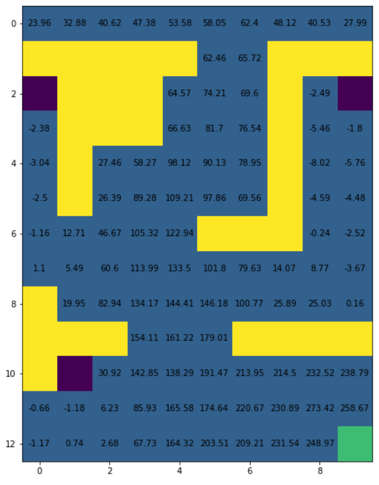
The MC Iterative learning algorithm was chosen due to there being the use of α to control the step size within iterations. Unlike the batch algorithm this is defined explicitly at the beginning of the solve function. This gives additional control for hyperparameter tuning. A total of 1000 episodes (also referred to as iterations or traces), as seen in the learning curves below, this is sufficient for the agent to learn enough to optimise its policy, and doesn't over compute. A value of $\alpha = 0.03$ was selected. Too low and the agent doesn't learn enough from episodes, too high and the solution does not converge well. ϵ was set to decay according to a power law, which struck the balance between exploration and exploitation of the value function. Section 2.5 details more on how these hyperparameters were chosen.

2.2 Plots of Optimal Policy and Value Function

In Figure 2 we can see the optimal policy and value function for the Monte Carlo control method. Whilst the general direction of travel according to the policy looks to arrive at the goal, in many locations the policy looks as if it is not the most efficient. We can put this down to insufficient exploration, as discussed in section 2.5. The values are much lower than those of the DP.



(a) MC Policy

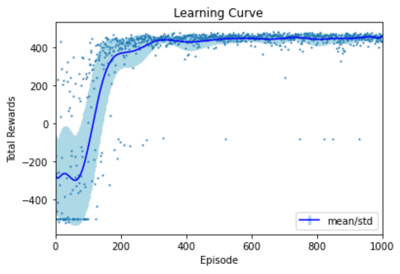


(b) MC Values

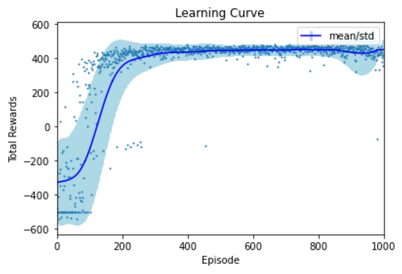
Figure 2: MC Policy and Value Plots

2.3 Replications

By replicating the Monte Carlo simulation and taking a mean of the resulting total rewards, we from Figure 3 can see that as we carry out more full simulations, we can reduce the noise around the data and fit a tighter curve to the results. A observation we can see for all three graphs is that at around the 300 episodes mark, the total rewards begin to stabilise at close to 400. Whilst we could fine tune the number of episodes to a smaller value still, this 'stable' range of data between 300-1000 episodes is useful for observations about the noise relating to number of replications. Clearly more simulations the better, however we can see that 15 replications in Figure 4 appears sufficient as the mean converges nicely, without too much difference from 40 replications.



(a) 3 Replications



(b) 40 Replications

Figure 3: Plots of mean and standard deviation for Monte Carlo learning curves

2.4 Learning Curve

In figure 4 we can see a learning curve for the Monte Carlo control. This shows as the agent undertakes more walks, the total rewards increases to a plateau around 400, indicating 'learning' and convergence to optimal policy.

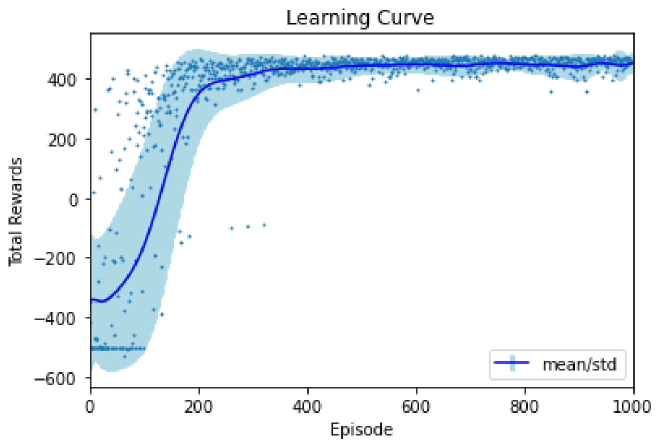


Figure 4: MC learning curve with total non-discounted rewards plotted against episode number, and the mean and STD curves for 15 replications

2.5 Exploring Parameters

Looking first at the variation of ϵ. This hyper-parameter will define that a policy is ϵ-greedy with respect to an action-value function estimate Q if for every state, with probability 1ϵ, the Agent chooses the greedy action, and with probability ϵ, the Agent selects an action uniformly at random from the set of available actions. Therefore the bigger we make ϵ, the more likely the agent is to pick one of the non-greedy actions. In order to guarantee that the MC control algorithm converges to the optimal policy, we need to ensure the conditions Greedy in

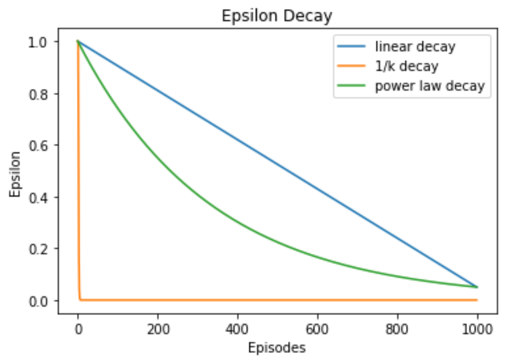
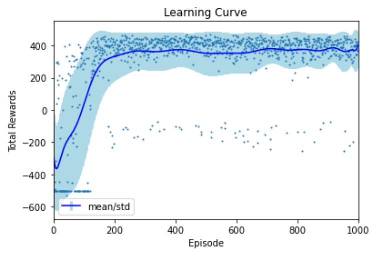
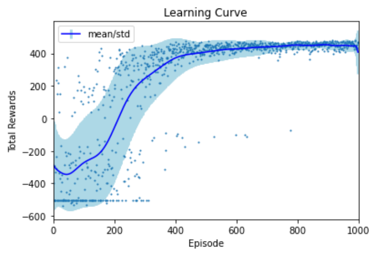


Figure 5: Various ϵ decay schedules

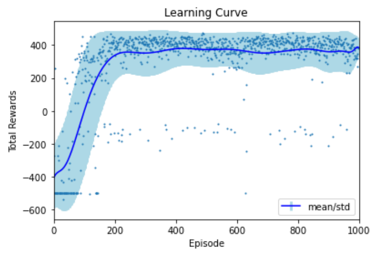
the Limit with Infinite Exploration (GLIE) are satisfied. We need to ensure the Agent continues to explore for all time steps, and the Agent gradually exploits more and explores less. The condition is satisfied therefore if we have some schedule for decay such that ϵ reduces with each step approaching zero. As explained in section 2.1, we selected a decay schedule according to a power law depreciation for ϵ. This reduces to a value very close to zero (0.05) as more steps are taken in Figure 5 below with two other decay schedule options. Immediately we see from the plot that for the '1/k' decay, i.e. ϵ is divided by the step number in the episode, the value for ϵ reduces drastically very quickly. This will have the effect of reducing the amount of exploration the agent undertakes. The agent will simply assign very high probabilities to the policy after a few episodes, but this policy doesn't have much information about what to do at many other state action pairs. The learning curve for this decay can be seen in Figure 6c), total rewards actually reach a high maximum early. However we can see the effects of the unexplored policy as the deviation of rewards remains high along the plateau, not fitting tight to the mean. This indicates that when an agent does get 'lost' from the optimal path it doesn't have sufficient information or ability to learn as epsilon has reduced so quickly. Figure 6a) below, shows the effect of choosing a constant ϵ value, which at first looks to be a good pass at converging to an optimum policy. However it does not satisfy the GLIE conditions and will therefore never converge as time progresses. The linear decay in Figure 6b) is improved but inferior to our power law learning curve.



(a) ϵ Constant



(b) ϵ Linearly decays



(c) ϵ Decays with 1/k

Figure 6: Variation of ϵ decay schedule on learning curve for total non-discounted rewards against episodes for Monte Carlo Control, using 20 replications

Selecting α strikes the balance between learning enough at each step (high α) and converging on an optimal policy (low α). To get close to the DP value function within 1000 episodes we need a high value but this means the policy will not converge to optimal. 0.03 was as high as could be chosen with a converging policy. Any higher and the solution doesn't converge, effectively learning too much from each step, updating the policy, when it might not be the right decision to make for the agent in that state action pair.

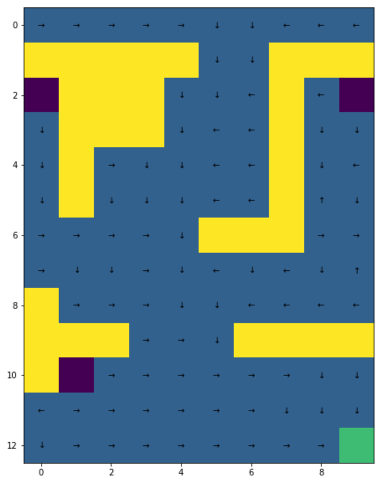
3 Question 3: Temporal Difference Reinforcement Learning

3.1 Method Justification

The SARSA method has been selected for this implementation of Temporal Difference Control. This is 'on-policy' TD control, with the alternative being Q-learning; a form of 'off policy' TD control. Neither one nor the other is necessarily better, if ϵ were gradually reduced then both methods would asymptotically converge to the optimal policy. Whilst the choice of method is based on the environment, there can be examples where SARSA can act to do more exploration and therefore give higher total rewards than Q learning; for this reason I chose it for my model. For the same reasons discussed in Question 2, I have chosen the same ϵ decay schedule (power law). I chose a value of α of 0.3 after some hyperparameter tuning. Once again this was the maximum value I could use for a converging policy.

3.2 Plots of Optimal Policy and Value Function

In Figure 7 we can see the optimal policy and value function for the SARSA Temporal Difference control method. We can see this policy looks highly effective for most points on the graph, with only a few locations where the policy takes you in a direction less efficient to that of the DP model. The values also reflect this whilst being significantly lower than the DP values.


(a) TD Policy

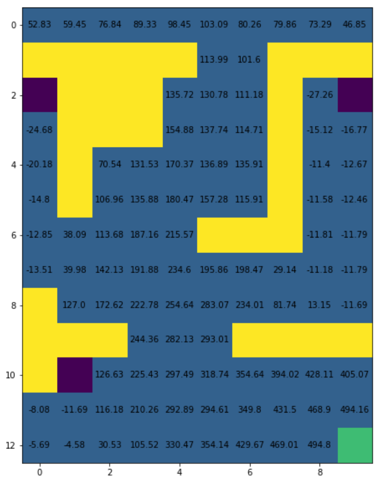

(b) TD Values

Figure 7: MC Policy and Value Plots

3.3 Learning Curve

In figure 8 we can see a learning curve for the Temporal Difference control. This shows as the agent undertakes more walks, the total rewards increases to a plateau around 400, indicating 'learning' and convergence to optimal policy.

3.4 Exploring Parameters

As seen in when exploring parameters in section 2, the way in which epsilon decays has a similar effect on the convergence of the optimal policy. Figure 9 demonstrates as much with the constant ϵ learning curve not converging well enough after 1000 episodes, the 1/k decay as converging quickly but not to an optimal policy, and the linear decay converging well but later than our best choice power law decay. This solution also satisfies the GLIE conditions. The choice of α is not as sensitive as that for the MC simulation. A value was chosen of 0.3, which is as high as it can be for the solution to converge to an optimal policy frequently returning high total rewards. Too high

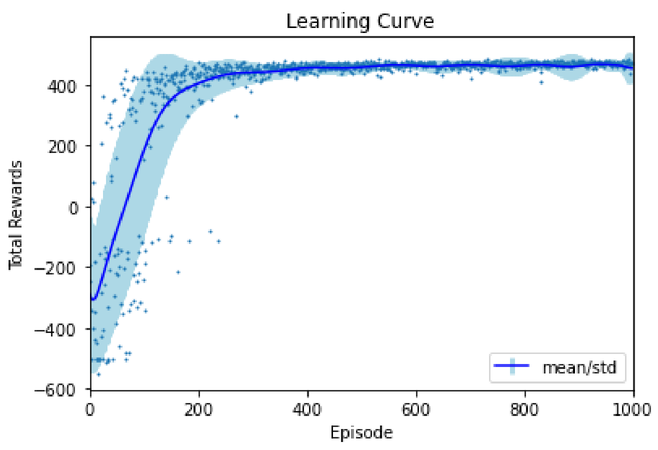
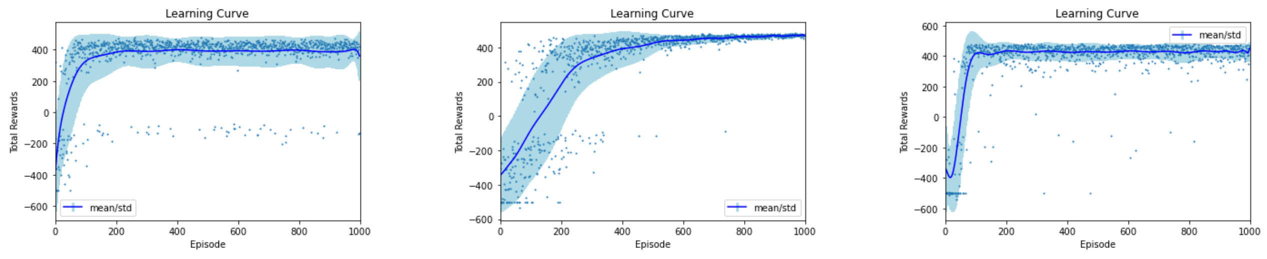


Figure 8: TD learning curve with total non-discounted rewards plotted against episode number, and the mean and STD curves for 15 replications and ϵ decay according to a power law

and the agent will learn too much from some experience without enough trials to back it up. As this is an order of magnitude higher than the α value for MC it leads to a closer replication of the DP value function.



(a) ϵ Constant (b) ϵ Linearly decays (c) ϵ Decays with 1/k

Figure 9: Variation of ϵ decay schedule on learning curve for total non-discounted rewards against episodes for SARSA Temporal Difference Control, using 20 replications

4 Question 4: Comparison

4.1 Value Function Estimation Error

The behaviour shown by by both models is as we would expect with the function estimation error starting high at the first episode and then reducing and leveling out for the final episode. The values for the mean squared error appear very high, but as seen from Figure 1, the value function plot for Dynamic Programming has very high values. This is related to the provided maze class gamma which was 0.98 for my unique CID, a lower gamma results in less information gain in the value iteration, and the values can end up much lower. We can see from the value plots for both MC and TD (in figures 2 and 7 respectively) by eye that the values are not closely aligned, and so Figure 10 gives us an indication of how the values line up relatively with the 'true' values from the DP value iteration. Both the MC and TD value error estimates level as they approach the optimal policy, even though this may not be considered the optimal value function. We can see the effects here of the ϵ decay according to the conditions of GLIE (see figure with constant eps)

There is a difference between the two control methods, as we see the MC mean level out higher to the temporal difference, some hyperparameter tuning shows that this can be reduced down by increasing the alpha value, but this also has the effect of destabilising the learning curve, with a less reliable total non-discounted rewards metric.

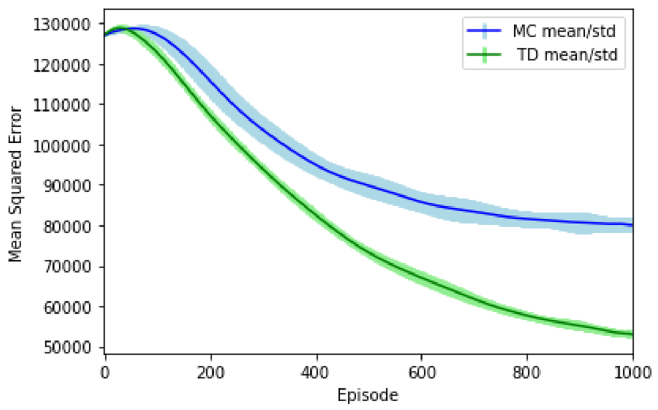
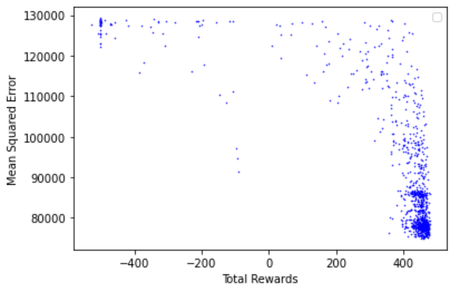


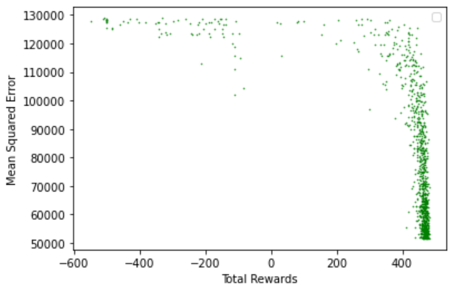
Figure 10: Mean squared error against number of episodes for both MC and TD control

4.2 Value Function Against Total Rewards

Both plots in Figure 11 show similar behaviour and the broad conclusion we can draw is that for any either control method, the less that the value function diverges from that of the dynamic programming function (in a relative sense), the higher the total non-discounted rewards will be. The relative nature of this is important. If I used a lower environment gamma value, this graph would look the same but with much different values for the mean squared error on the y axis, with the scatter plot essentially 'shifted' down. This does not mean that my solution or techniques for MC/TD are any less valid as the rewards still group as being near maximum as the optimum policy converges. The plots tell us that the value function is not that important when evaluating a policy or learning method.



(a) Comp/Monte Carlo Control



(b) Comp/Temporal Difference Control

Figure 11: MSE against total non discounted rewards