Imperial College London

COURSEWORK 1

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

# COMP97143: Reinforcement Learning

*Author:*
Jaime Sabal Bérmudez (CID: 01520988)

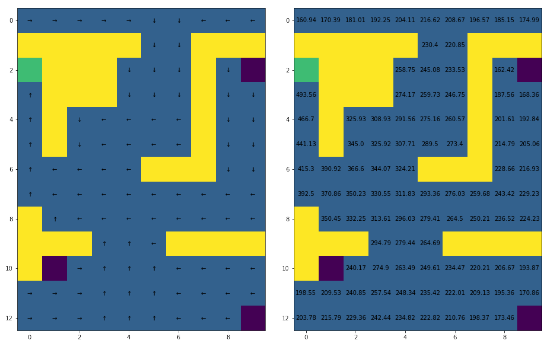
Date: November 8, 2021

## Question 1: Dynamic Programming

### 1.1: Method Chosen to Solve Grid-World Problem

The method chosen to solve this environment using Dynamic Programming (DP) was value iteration. This method builds upon the ideas of policy evaluation and policy iteration but differs from them in the manner in which the optimum policy is found. In value iteration, the agent performs policy evaluation only once at each step of the learning and estimates the value $V(s)$ of the state $s$ to be $\max_a Q(s,a)$. This is repeated at every step until $V(s) \to V^*(s)$, where $V^*(s)$ is the optimum state value function. This is in contrast to policy iteration, where policy evaluation is performed until the optimum policy is found and consequently updated. After experimenting with both methods, I found that value iteration arrived to $V^*(s)$ and hence the optimum policy around five times faster than policy iteration in this specific environment. Finally, we set the condition for convergence to be $\max_s |V_k(s) - V_{k-1}(s)| < 0.0001$, with $s \in \mathcal{S}$, $\mathcal{S}$ being the set of states in the environment.

### 1.2: Graphical Representation of the Optimal Policy and Value Function



**(a)** Optimal policy. **(b)** Optimal value function.

**Figure 1:** Optimal policy and value function using the CID-personalised parameters $\gamma = 0.96$ and $p = 0.82$.

### 1.3: The effect of $\gamma$ and $p$ on the Optimum Policy and Value Function

When using the CID-personalised discount factor $\gamma = 0.96$ and varying the probability of success $p$, interesting properties of the environment dynamics are revealed. When $p < 0.25$, the agent will actually be more likely to take actions that don't maximise $Q(s,a)$, since these will have a larger combined probability of success than the action that does. Therefore, the policy will converge to one that leads to the terminal states with low rewards, rather than the goal state with the highest. On the other hand, using $p = 0.25$ has the effect of inhabilitating the convergence to the optimal policy, since the agent will always choose an action randomly regardless of its effect on the future rewards acquired. Finally, $p > 0.25$ allows for convergence to occur, but $p$ must be somewhere in between 0.25 and 1 such that the greedy policy still has a chance of not being carried out for any state $s$.

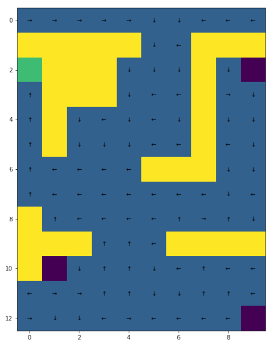
Varying $\gamma$ whilst $p = 0.82$, on the other hand, has the effect of changing the sensibility of our agents' decision process to the sum of all future rewards when following the current policy $\pi(s)$. If $\gamma < 0.5$, the agent will update the value $V(s)$ of a given state $s$ based on the rewards obtained in the more immediate future by following $\pi(s)$. The agent will then be unable to learn the optimal policy for states that are far away from the goal state. If $\gamma > 0.5$, future rewards will be taken more into account and the optimal policy is more likely to be found in all the states. Nevertheless, it is also not optimal that $\gamma = 1$, since a lot of the actions that can be taken from a state don't have long-lasting repercussions and thus a lot of useless information will be taken into account in the learning process.

## Question 2: Monte-Carlo Reinforcement Learning

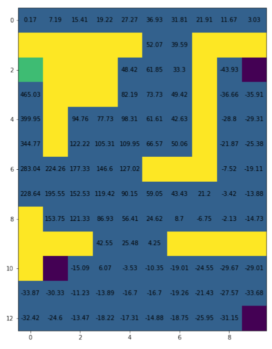
### 2.1: Method Chosen to Solve Grid-World Problem

The method chosen to solve the grid-world problem for the Monte-Carlo agent was an on-policy first-visit control algorithm with $\epsilon$-soft policies, meaning the agent always has a finite probability of exploring states that offer lower perceived rewards. This is important since the agent must learn from sampled experience, and the optimum policy at each state $s$ can only be learned if its value is sampled enough times. Moreover, we use an on-policy approach due to the fact that we want the agent to act upon the epsilon-greedy policy updated throughout learning. However, we make sure to initialise epsilon to a number close to 1 (namely, $\epsilon = 0.95$) so that in the earlier episodes the states that don't maximise the state-action value function are prioritised. After every episode, we then reduce $\epsilon$ by a constant decay factor $\beta = 0.999$ to ensure that our policy is greedy in the limit of infinite exploration (GLIE). Moreover, we retrieve the total discounted reward $G(s,a)$ from the first visit to each state-action pair in the episode, and then update $Q(s,a)$ by averaging $G(s,a)$ over the total number of occurrences $N(s,a)$ for $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$ throughout all episodes (discussed this further in section 2.5). Finally, we use 1,000 as the number of episodes in learning because a policy reasonably close to the optimal one is achieved in this time.

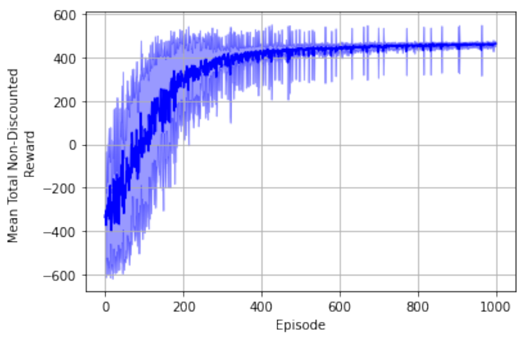
### 2.2-2.4: Graphical Representation of the Optimal Policy and Value Function, and the Learning Curve of Agent



**(a)** Optimal policy

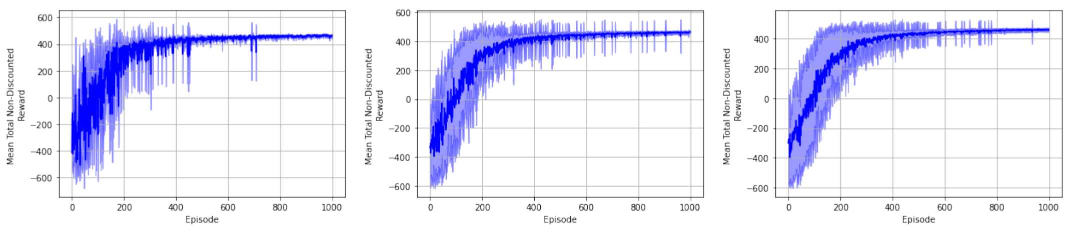


**(b)** Optimal value function



**(c)** Average learning curve

**Figure 2:** Optimal policy and value function (**a)** and **b)**, respectively); **c)** shows the average learning curve across 25 replications for 1,000 episodes, (shaded area represents the standard deviation) using a starting epsilon $\epsilon = 0.95$ and $\beta = 0.999$.



**(a)** 5 **(b)** 25 **(c)** 50

**Figure 3:** Average learning curves for different numbers of replications on the MC agent: **a)** 5 replications ; **b)** 25 replications ; **c)** 50 replications.

Figure 3 shows the average learning curves for a different number of learning replications produced. It is important to find a sufficient number of replications needed to get a sense of the average results produced in the MC learning process due to variability of these arising from the nature of the process. Specifically, starting off with $\epsilon$ close to 1 implies that the actions taken will be nearly random in the earlier episodes, meaning that different runs will have very different total non-discounted rewards for the first few hundred episodes. By replicating the process, we want to get a sense of the expected value of this variance between difference runs. By the law of large numbers, the standard deviation shown by the shaded area around the learning curves in Figure 3 will become perfectly smooth for an infinite number of replications. However, and due to time constraints, it looks as though 25 replications is a sufficient number from the little improvement seen between Figure 3b and Figure 3c.

## 2.5 Effect of $\epsilon$ and $\alpha$ on the Learning Curve of the Agent

For an on-policy first-visit MC control algorithm with $\epsilon$-soft policies, the exploration parameter $\epsilon$ determines the probability of an action $a$ being taken from a given state $s$ through the policy $\pi$ update:

$$\pi(a|s) \leftarrow \begin{cases} 1 - \epsilon + \epsilon/|\mathcal{A}(s)| & \text{if} \quad a = a^* \\ \epsilon/|\mathcal{A}(s)| & \text{if} \quad a \neq a^* \end{cases} \tag{1}$$

where $a^* = \text{argmax}_a Q(s,a)$ and $\mathcal{A}(s)$ is the set of actions the agent can take in the given state $s$.

We want to choose an $\epsilon$ that shows an average total non-discounted reward with a large but stable standard deviation for the earlier episodes, indicating that it is doing a lot of exploring (i.e the policy prioritises the actions $a \neq \text{argmax}_a Q(s,a)$ for $a \in \mathcal{A}(s)$). However, we also want to choose an $\epsilon$ that shows a decreasing standard deviation as the learning progresses, since this would imply that the agent has found the optimum policy from all starting states and is obtaining rewards that converge to those obtained using a DP approach. This can be achieved by simply multiplying $\epsilon$ after every episode by a constant decay factor $\beta$, with $0 < \beta < 1$ (GLIE).

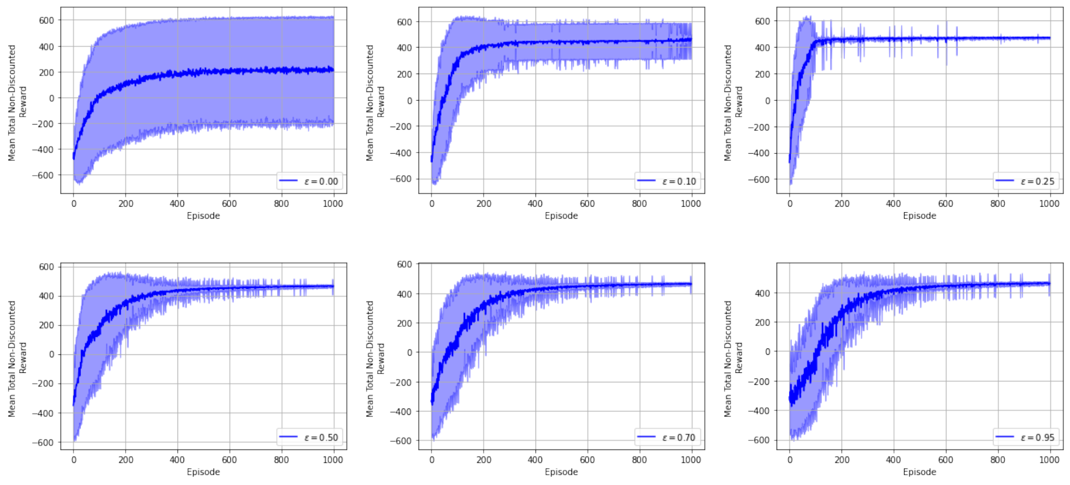


**Figure 4:** Average total non-discounted rewards across 25 replications for different starting values of epsilon when reducing it by a factor $\beta = 0.999$ after each episode for 1,000 episodes.

From the plots shown in Figure 4, it seems that it is best to use large starting values for $\epsilon$ since these align better with the aforementioned desired characteristics. In the end, we settle on an optimum value of $\epsilon = 0.95$ (shows a more stable $\sigma$ than $\epsilon = 1.0$).

In terms of $\beta$, we want that the agent does a lot of exploring earlier in the learning process and that in the later stages it becomes a fully greedy policy (best action found is assigned a probability of practically 1 of occurring, i.e epsilon converges to 0). I found that $\beta = 0.999$ allows for this to occur nicely and, in 1000 episodes, our agent seems to always end up in the goal state when reset to any one of the defined starting states ($0.999^{1000} = 0.368$, meaning by the end of the learning $\epsilon$ will have reduced by a factor of 0.368). Nevertheless, it is always important to consider the pay-off in terms of computational effort in using smaller values for this parameter; for values approximating 1, the duration of each episode at the beginning of the learning process is very large since the agent explores a lot and therefore exhausts the set limit of 500 steps much more often. However, for this environment (which is quite small), reducing $\epsilon$ by a constant factor close to 1 seemed appropriate, since the agent consistently found a policy that very close to its optimum in a reasonable amount of time.

For our chosen MC control algorithm $\alpha$ can be interpreted as being equal to $1/N(s,a)$, since we are updating $Q(s,a)$ through:

$$Q(s,a) \leftarrow \frac{1}{N} \Sigma_{i=1}^{N} R_i(s,a) \tag{2}$$

where $R_i(s,a)$ is the return of the state-action pair $(s,a)$ in the episode $i$ out of the $N$ episodes in which it has appeared. We choose this $\alpha$ because it is an efficient way of updating $Q(s,a)$ without having to keep track of all the sampled returns for each state-action pair.

## Question 3: Temporal Difference Reinforcement Learning

### 3.1: Method Chosen to Solve Grid-World Problem

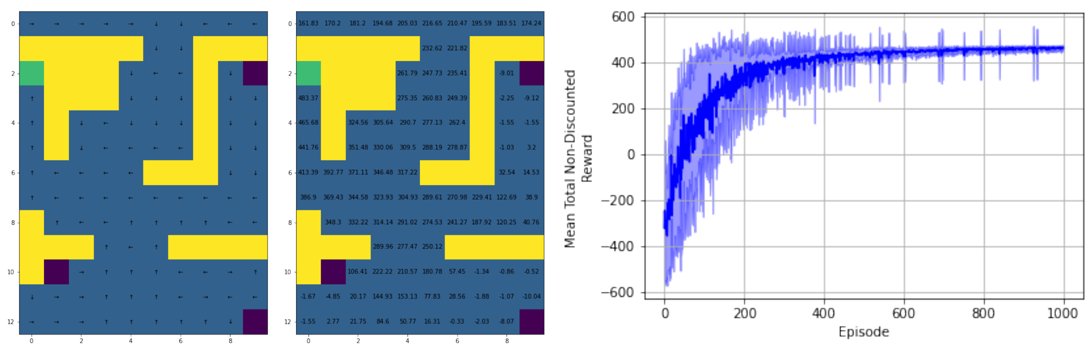
The method chosen to solve the grid-world problem using Temporal-Difference was Q-learning. This is an off-policy algorithm that calculates $Q(s,a)$, with the action a being chosen through an $\epsilon$-greedy policy, by assuming that a fully greedy policy is followed in the next state $s'$ through the update:

$$Q(s,a) \leftarrow (1-\alpha)\,Q(s,a) + \alpha\,(\mathrm{r} + \gamma \max_{a'} Q(s',a')) \tag{3}$$

where $\alpha$ is the learning rate, $\gamma$ is the discount factor, and $a' \in \mathcal{A}(\mathrm{s}')$.

The advantage of using an off-policy method is that the agent is able to freely take exploratory actions while an $\epsilon$-greedy policy is being evaluated. If we reduce $\epsilon$ by a constant decay factor $\beta$ like we did for the MC control method, the value function is ensured to converge to its optimum $V^*$ in the limit of infinite exploration. Thus, and for the same reasons as for MC learning, we choose $\epsilon = 0.95$ and $\beta = 0.999$. On the other hand, we choose a learning rate $\alpha = 0.3$ after investigating the effects of choosing a range of different values between 0 and 1 in section 3.4. Finally, we use the same number of episodes as in MC learning for the purpose of comparing the two in question 4, even though it is clear that convergence is achieved much faster than that (in $\sim 300$ episodes).

### 3.2 - 3.3: Graphical Representation of the Optimal Policy and Value Function, and the Learning Curve of Agent



**(a)** Optimal policy **(b)** Optimal value function **(c)** Average learning curve

**Figure 5:** Optimal policy and value function (**a)** and **b)**, respectively); **c)** shows the average learning curve across 25 replications for $1,000$ episodes, (shaded area represents the standard deviation) using a starting epsilon $\epsilon = 0.95$ and $\beta = 0.999$.

### 3.4: Effect of $\epsilon$ and $\alpha$ on the Learning Curves of the Agent

After computing the learning curves for a constant $\alpha$ and varying $\epsilon$ for the TD agent, the conclusions drawn were analogous to those in section 2.5 for the MC agent (see Figure 6). This is due to the fact that the objective remains that the agent should explore a lot in the beginning of the learning process and by the end follow a considerably more greedy policy ($\epsilon$ reduced by a factor of 0.368 by the last episode, as with the MC agent).

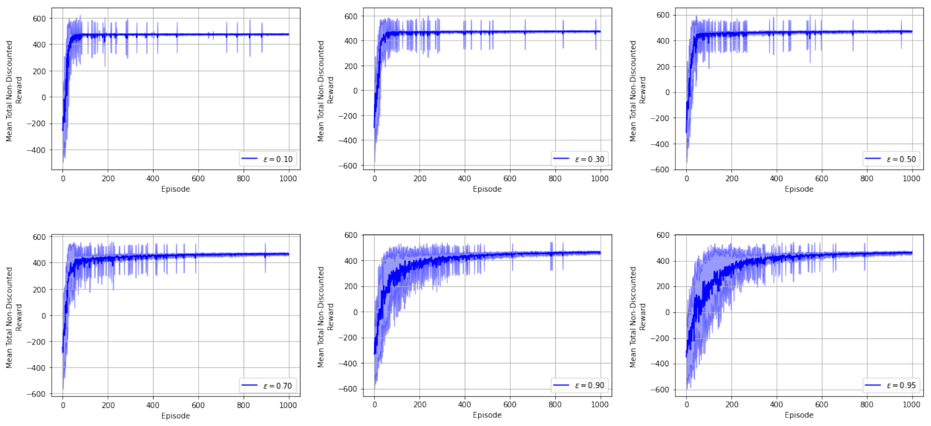


**Figure 6:** Average total non-discounted rewards across 25 replications for different starting values of epsilon when reducing it by a constant factor of 0.999 (GLIE) after each episode for 1,000 episodes and using a learning rate $\alpha = 0.3$.

In terms of the optimum value for $\alpha$, in Figure 7 we can see the average learning curves across 25 replications for different values of $\alpha$ using $\epsilon = 0.95$ and $\beta = 0.999$. Clearly, for $\alpha = 0.0$ the learning curve will be flat since the agent won't be changing $Q(s,a)$ at any step of the learning process. On the other hand, as $\alpha$ is gradually decreased from 1 to 0, the learning curves are seen to become smoother and with a lower variance in the final episodes, albeit all appear to converge to the optimal total non-discounted rewards. This makes sense, since a lower learning rate implies less drastic changes in updating $Q(s,a)$ (less weight given to $\max_{a'} Q(s',a')$ within the update in Eq. 3). For this reason, we argue that a value of $\alpha = 0.3$ will allow the TD agent to converge nicely to the optimum value function V*.

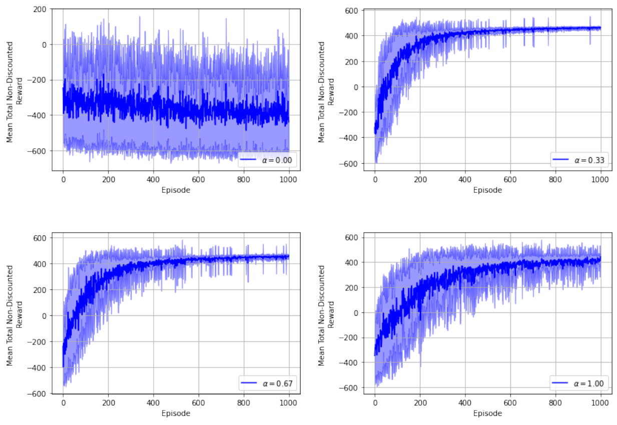
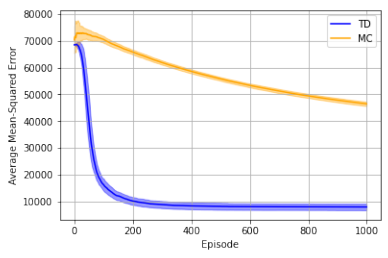


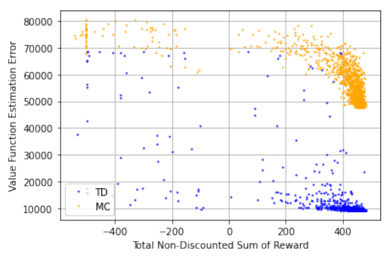
**Figure 7:** Average total non-discounted rewards across 25 replications for different learning rates $\alpha$ with $\epsilon = 0.95$, $\beta = 0.999$, episodes = 1000.

## Question 4: Comparison of Learners

### 4.1, 4.3:



**(a)** Estimation error of learning agents



**(b)** MSE against total non-discounted rewards

**Figure 8: a).** Value function estimation error across 25 replications for both MC and TD learners using the optimum parameters found in questions 2 and 3 (shaded areas show standard deviation for each learner); **b).** Estimation error against total rewards for one run of the learning process for both agents.

### 4.2: Analysis of Figure 8a

From Figure 8a it is clear that the TD agent converges much quicker to the optimum value function, since the value-function estimation error has a much steeper gradient in the first $\sim$ 200 episodes and seems to stabilise after that at a mean-squared error (MSE) of around 10,000, while for the MC agent it it only reaches an MSE of around 45,000 by the last episode. This occurs because the TD agent retrieves the value of a state directly from the estimated value of the subsequent state (bootstrapping), while the MC agent updates the $\epsilon$-greedy policy (which it uses to evaluate the state-action value function of any given state) only once per episode, resulting in much slower learning. Finally, it is important to note how the MSE values are surprisingly large for both agents (especially MC), which is a direct result of the combination of large CID-personalised values of $\gamma = 0.96$ and $p = 0.82$. Specifically, having a large value for $p$ doesn't allow for much deviation from the $\epsilon$-greedy policy, which causes the agent to only explore the high-rewarding states.

### 4.4: Analysis of Figure 8b

In Figure 8b one can clearly see that the TD learning method presents a much lower variance than the MC method. The bulk of points for the TD agent have a high total non-discounted reward and a low MSE, meaning the estimation error is very low (i.e value function very similar to that of the DP agent) and the agent consistently obtains high rewards (i.e optimal policy is acquired and followed in the final 80% of the episodes). In contrast, the bulk of points for the MC agent have a high MSE and a high total non-discounted reward. This implies that the value function does not converge to its optimal form and is quite far off from it for most states in the environment (see Figures 1b and 2b). Based on this, it is clear that it is not important to have a good estimation error in order to have a good reward, since the MC agent just learns the optimal policy for the states that are between the starting states and the goal state and follows that consistently to do so while presenting a bad estimation of the values of the other states.