AI and Machine Learning Coursework

Sebastian Lynch

March 13, 2025

1 Task 2 - Reinforcement Learning in Gridworld

1.1 Introduction

Reinforcement learning (RL) is a subset of machine learning focused on how to map situations to actions, so as to maximise a numerical reward signal [1]. It boasts a wide range of real-world applications such as in industrial automation and mastering board games. A famous example comes from the board game Go, where the program, AlphaGo, defeated one of the greatest Go players, Lee Sedol, who has won eighteen world titles [2].

Gridworld is an environment for testing RL algorithms, where an agent navigates a maze-like grid, avoiding obstacles and reaching a goal with a high reward. This is visualised as a mouse (agent) navigating the grid which contains walls, spikes (negative reward), and cheese (reward). The environment is displayed in Figure 1.

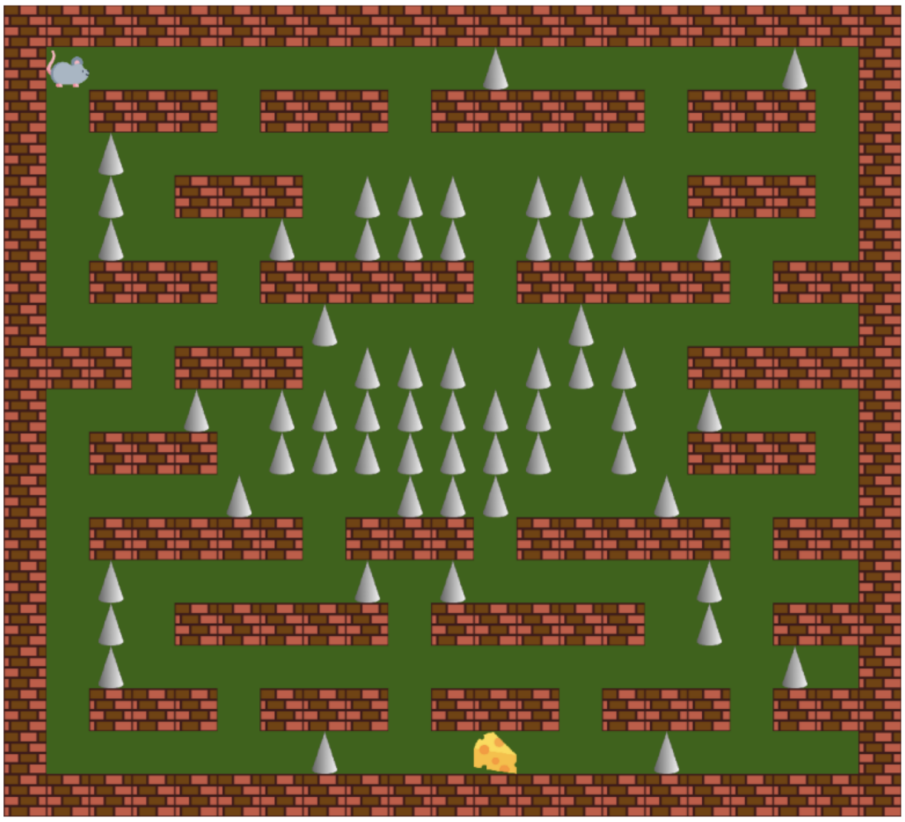

Figure 1: The specified learning environment.

To train the agent, we will implement and compare two RL algorithms:

- Value Iteration – An approach that iteratively updates the value of each state based on future rewards.
- Q-Learning – An algorithm that learns action-value functions through exploration and reward feedback.

We will evaluate how well each method helps the agent learn optimal policies for navigating the maze. A policy is a mapping that guides the agent's actions by determining which action to take in each given state. An optimal policy is the strategy that maximises the agent's expected cumulative reward over time. Finally, we analyse how hyperparameters influence Q-Learning's performance.

1.2 Value Iteration

1.2.1 Explanation

The goal of value iteration is to compute the optimal value function for each state in a Markov Decision Process (MDP). An MDP is a fully observable, probabilistic state model. value iteration finds the optimal value function by solving the Bellman equations iteratively. The Bellman equation describes the relationship between the value of a state and the values of its neighboring states.

$$V(s) = \overbrace{\max_{a \in A(s)}}^{\text{best action from } s} \overbrace{\underbrace{\sum_{s' \in S}}_{\text{for every state}} P_a(s' \mid s)[\ \underbrace{r(s,a,s')}_{\text{immediate reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{V(s')}_{\text{value of } s'}\ ]}^{\text{expected reward of executing action } a \text{ in state } s}$$

Figure 2: The Bellman equation, [Day 7 lecture notes].

Here, s is the current state, s' is the next state (after taking action a), and $P_a(s'|s)$ is the probability of transitioning from state s to state s' when action a is taken.

By iterating over the states and updating their value estimates, value iteration converges to the optimal policy. The iterative process is shown below.

Initialise $V_0(s)$ arbitrarily $\forall s \in S$ and $i = 0$

Repeat

$\Delta \leftarrow 0$

For each $s \in S$

$V_{i+1}(s) \leftarrow \max_{a \in A} \sum_{s' \in S} P(s'|s,a)[R(s,a,s') + \gamma V_i(s')]$

$\Delta \leftarrow \max(\Delta, |V_{i+1}(s) - V_i(s)|)$

$i = i + 1$

Until $\Delta \leq \epsilon$

Firstly, we initialise both $V_0(s)$ arbitrarily for all states, and $i = 0$; i is the iteration counter. We then move into the iterative improvement loop. We set delta (Δ) to 0, then, for every state, s in the set of all states, S, we update the value function. We also track the largest change, setting it equal to Δ. The repeat loop ends when the value function's largest change is below a small threshold, ϵ, indicating convergence.

This algorithm provides us with the optimal state-value function $V^*(s)$, which represents the maximum expected return for each state s. This is used to derive the optimal policy $\pi^*(s)$:

$$\pi^*(s) = \arg\max_{a \in A} \sum_{s \in S} P(s'|s,a)[R(s,a,s') + \gamma V*(s')].$$

1.2.2 Implementation

This algorithm has 2 hyperparameters. The discount factor, γ, determines the importance of future rewards. A higher value means that the agent values long-term rewards more. The range is $0 \leq \gamma \leq 1$. The other hyperparameter is the convergence threshold, ϵ. This value specifies at which point the algorithm stops iterating (when $\Delta \leq \epsilon$). For our value iteration model, we use an ϵ value of 10^{-7} and a γ value of 0.7.

The visual representation of the optimal policy derived is shown by the arrows in Figure 3.

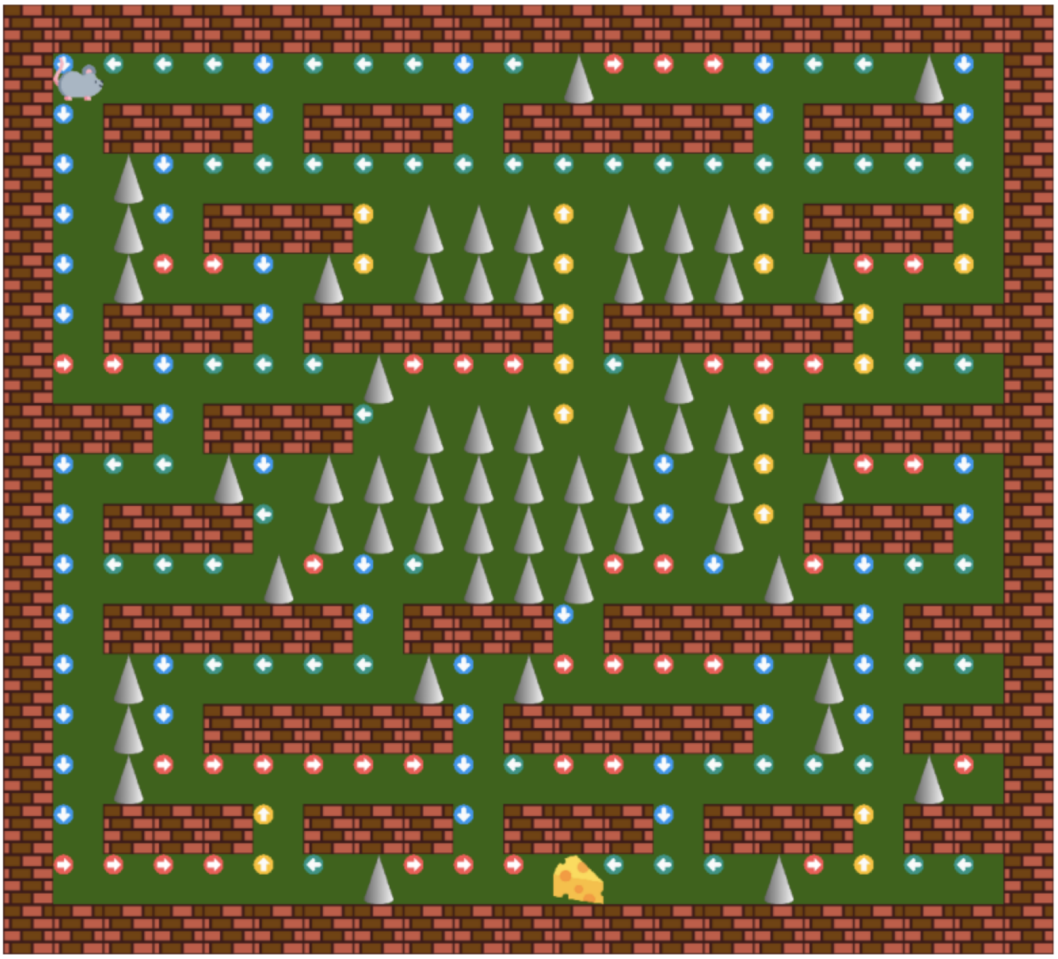

Figure 3: The value iteration policy mapping in our environment.

The agent, who starts in the top left corner, has found the optimal path to the reward (cheese). All of the arrows (policy) correctly guide the agent toward the cheese, except for those in states entirely enclosed by walls or spikes, where escape is impossible. The program runs in under 1 second.

1.3 Q-Learning

1.3.1 Explanation

Q-learning is an algorithm that estimates the value of actions in specific states to optimise decision-making. This is achieved by learning the Q-values (state-action values), which estimate the long-term reward for taking an action in a given state. It is model-free as it does not require a model of the

environment. The Q-values are given by:

$$Q(s,a) = \sum_{s' \in S} P(s'|s,a)[R(s,a,s') + \gamma V(s')].$$

The Q-learning algorithm is shown below, from [3].

Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Repeat (for each step of episode):
    Choose a from s using policy derived from Q
    Take action a, observe r, s'
    Update
      $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
    $s \leftarrow s'$;
  Until s is terminal

We begin by initialising the Q-table (Q(s, a) with all state-action pairs) arbitrarily. An episode is a complete sequence of interactions between an agent and the environment, starting from the initial state and ending in a terminal state. The number of episodes is a hyperparameter. For each episode, we initialise s, this means placing the agent in an initial state (the top left corner of the grid in our case). Then, for each step of an episode, we select an action a from the current state s. This action is chosen using a policy derived from Q. We use an epsilon-greedy exploration strategy; meaning the agent takes a random action with probability ϵ, where ϵ is a hyperparameter. After choosing and subsequently taking the action, we observe the immediate reward, r and the next state, s'. We then update the Q-table using the equation shown, where α is a hyperparameter representing the learning rate. This is repeated until s is terminal. After the specified number of episodes, the policy is extracted using:

$$\pi^*(s) = \arg\max_{a \in A} Q(s, a).$$

1.3.2 Implementation

There are 4 hyperparameters for the Q-learning model:

- Learning rate, α determines how much newly acquired information overrides the existing knowledge. We choose it to be 0.3, meaning that 30% of the new Q-value is incorporated.
- Discount factor, γ, controls how much the agent values future and immediate rewards. We choose a value of 0.9, meaning that the agent heavily considers future rewards.
- Exploration rate, ϵ, dictates how often the agent takes random actions instead of following the best-known policy. We choose a value of 0.4, meaning that the agent explores 40% of the time, using its learned knowledge 60% of the time.
- Episodes, which we choose a value of 10,000 for. This ensures that the agent has ample opportunities to explore, refine its policy, and converge to an optimal solution.

The visual representation of the optimal policy derived from Q-learning is shown by the arrows in Figure 4.

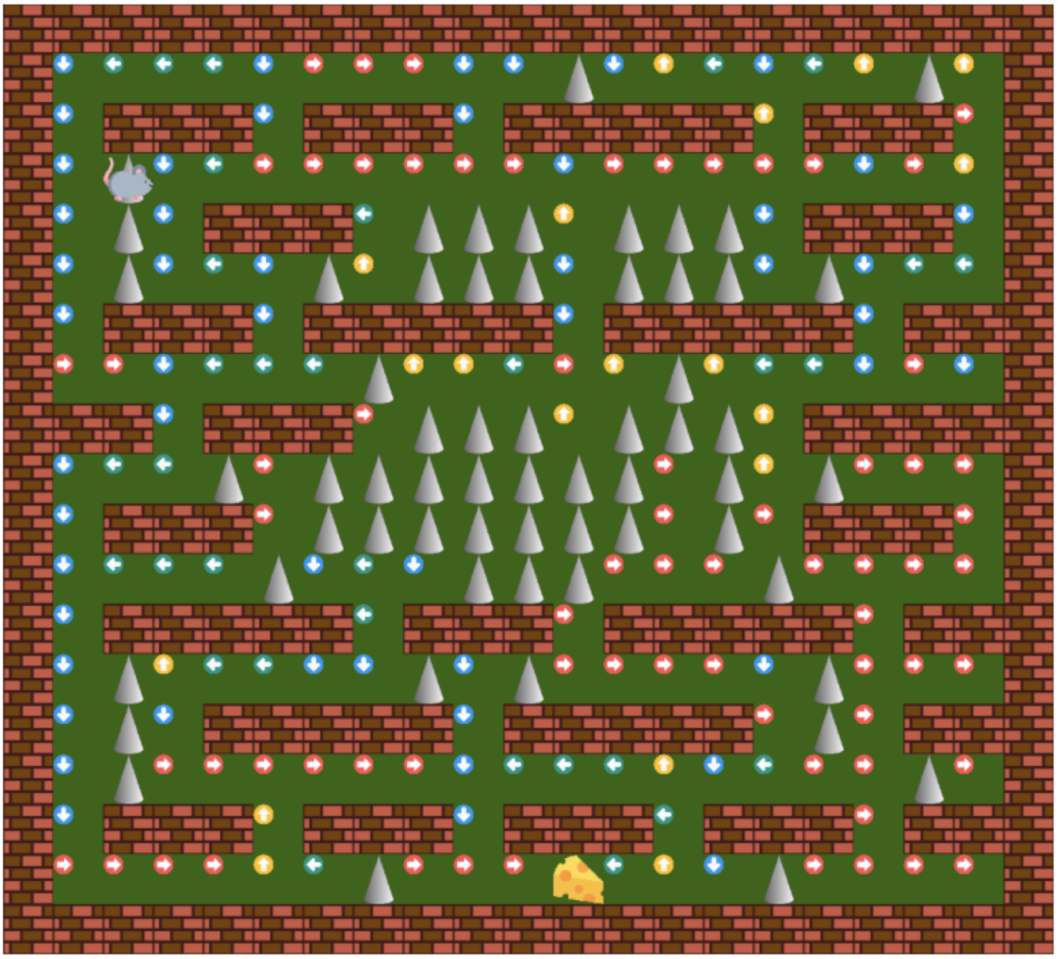

Figure 4: The Q-learning policy mapping in our environment.

The agent again found the optimal path to the reward from its starting point, however, many of the arrows away from the optimal route do not point in logical directions. After reviewing the Q table, it is clear that many states were never reached by the agent, most noticeably in the bottom right corner. The program took 48 seconds to run.

1.4 Comparing Value Iteration and Q-Learning

Firstly, the value iteration algorithm ran significantly quicker than the Q-learning algorithm, although Q-learning still ran in under 1 minute. Both algorithms found the optimal (shortest) path from the starting point to the reward. For the value iteration algorithm, all states which are accessible have a policy correctly guiding the agent toward the cheese. Meanwhile, many states are never reached by the agent in Q-learning. This occurs because the Q-learning algorithm starts from the top left and explores from there, whereas the value iteration algorithm iterates over all states. It may be possible to correct the Q-learning's shortcomings by adjusting the discount factor and exploration rate.

Another way to compare these models is to plot the average reward that each model produces. This plot is shown in Figure 5.

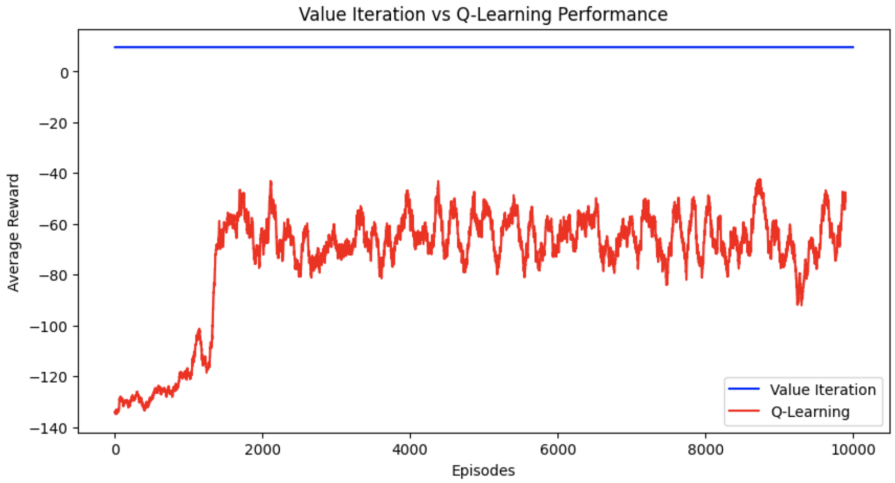

Figure 5: Average reward for each model.

This demonstrates value iteration's superiority in this specific task. The Q-learning algorithm quickly plateaus at an average reward of around -60, whilst the value iteration's is positive.

1.5 Exploring Discount Factor and Exploration Rate of Q-learning

First, we try increasing the exploration rate, from 0.4 to 0.7. The resulting policy is shown in Figure 6.

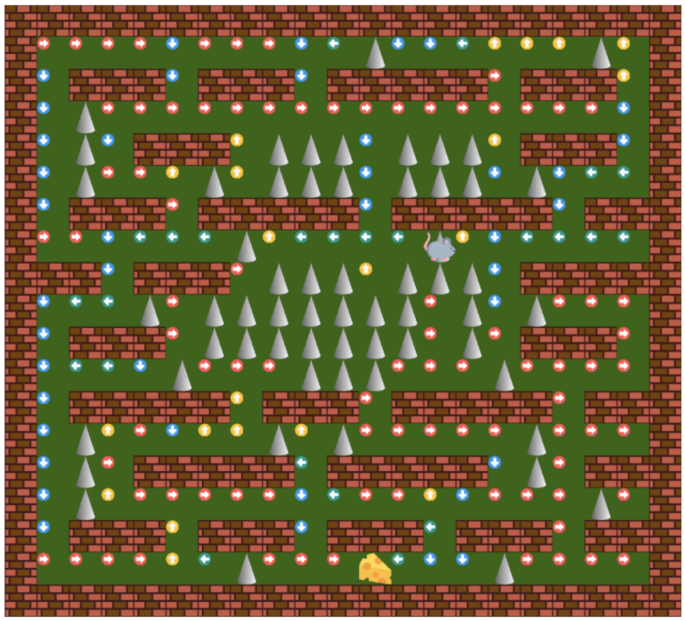

Figure 6: The Q-learning with a higher exploration rate policy mapping in our environment.

This change has not produced the desired effect; the agent no longer follows a path from the starting position to the reward. The problem of the agent never reaching a large number of states remains.

Next, we reset the exploration rate back to 0.4 and try adjusting the discount factor. We reduce the value from 0.9 down to 0.7. The resulting policy is shown in Figure 7. This change also has not produced the desired effect, although the agent still has the optimal path from the starting position to the reward. The problem of the agent never reaching a large number of states remains.

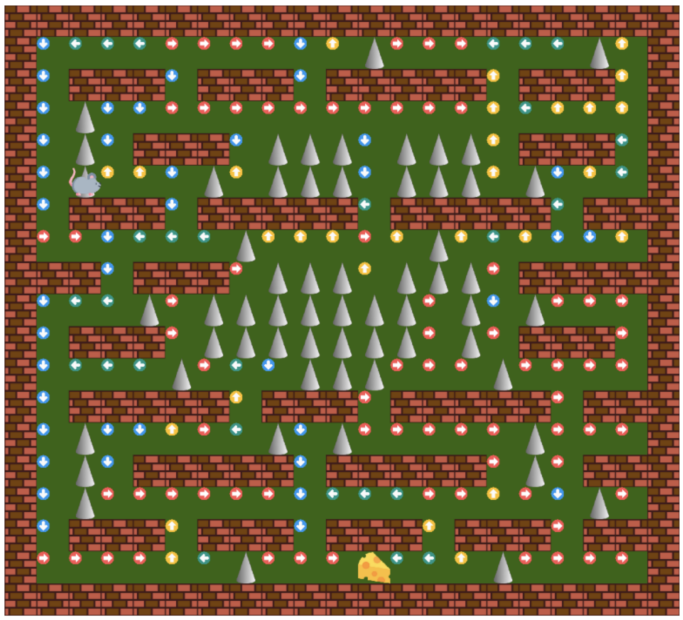

Figure 7: The Q-learning with a lower discount rate policy mapping in our environment.

Another option is to decrease the exploration rate over time. To do this, we set the initial exploration rate to 0.8 with a decay rate of 0.995. The exploration rate is multiplied by the decay rate at the end of each episode. The minimum exploration rate value is 0.1; once this value is reached, the decay rate is no longer applied. The resulting policy is shown in Figure 8.

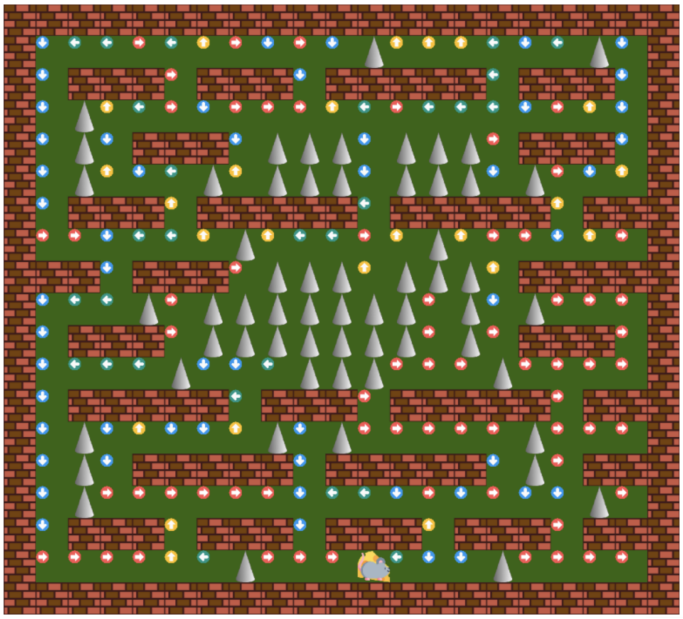

Figure 8: The Q-learning with a decaying exploration rate policy mapping in our environment.

There are some improvements here; some of the bottom right area is explored and the optimal path remains. Despite the improvement, a large portion of the grid still remains unexplored. Using this method, the average reward shows a huge improvement, shown in Figure 9.

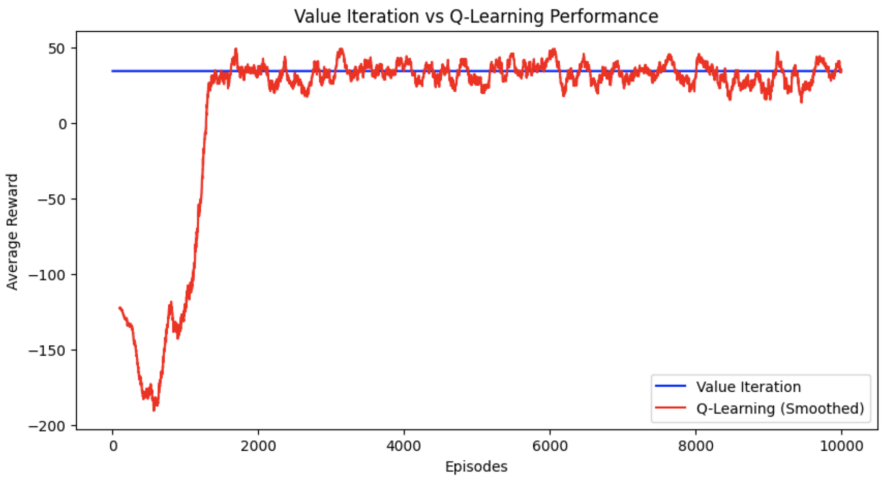

Figure 9: Average reward for each model, value iteration and the Q-learning with decaying exploration rate.

1.6 Conclusions

Value iteration is the clear winner for this specific task; it runs quickly and the policy is correctly optimised for each state. The Q-learning method does find the optimal path to the reward from the start position; however, it takes longer to run and does not explore a number of states, which happens as the agent starts in the top left corner and does not iterate over all states. The Q-learning's performance does improve when applying a decaying exploration rate.

Since Q-learning is model-free, it is more scalable and therefore better suited for larger mazes than the one we used. It is also better for dynamic environments where the transition model is unknown or changes over time (eg. slip rate is non-zero).

1.7 Future Work

With more time, further experimenting of the discount and exploration rates in Q-learning could reveal if the agent can match value iteration. Additionally, testing on a larger maze could highlight Q-learning's strengths.

References

[1] Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction*. Vol. 1. 1. MIT press Cambridge, 1998.

[2] Sean D Holcomb et al. "Overview on deepmind and its alphago zero ai". In: *Proceedings of the 2018 international conference on big data and education*. 2018, pp. 67–71.

[3] Kao-Shing Hwang et al. "Cooperation Between Multiple Agents Based on Partially Sharing Policy". In: vol. 4681. Aug. 2007, pp. 422–432. ISBN: 978-3-540-74170-1. DOI: 10.1109/ICSMC.2007.4413575.