SC4003 Assignment 1

Koh Ming En

March 3, 2025

1 Introduction

This report will discuss Value Iteration and Policy Iteration, which are dynamic programs that compute optimal values for solving Markov Decision Processes (MDPs). To demonstrate the capabilities of both Value Iteration and Policy Iteration, we will test both algorithms in a Maze environment as shown in Figure 1. Section 2 will cover the overview of the algorithms, the resulting optimum policy and the estimated utilities of each state. In this project, the agent will only consider actions that will lead the agent to in-bound regions and if the action does lead to an out-of-bound region, the agent will simply stay in place. This rule was implemented to ensure that the algorithm will provide an optimum policy with valid actions only.

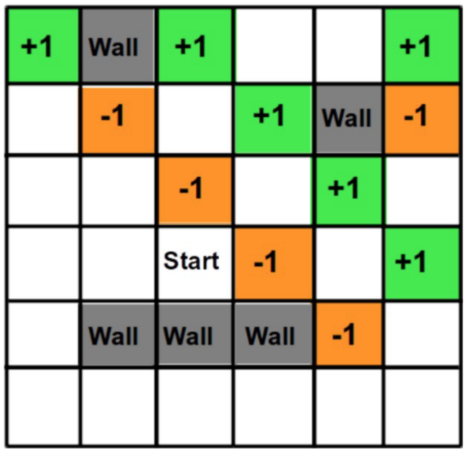


Figure 1: Maze used in Section 2

In section 3, we will test the algorithms on a more complex maze environment, where the agent is supposed to find the exit of a maze with multiple dead-ends. We will also discuss the time complexity of the two algorithms to estimate an upper bound on the complexity of the environment for these algorithms to terminate.

This project was coded in Python, with the `Matplotlib` library used for plotting graphs and `tkinter` library used for visualizing the estimated utilities and optimum policies.

2 Solving the Maze

2.1 Value Iteration

2.1.1 Description

Value Iteration is a dynamic-programming method for finding the optimal value function V^* by solving the Bellman equations iteratively. It uses the concept of dynamic programming to maintain a value function V that approximates the optimal value function V^*, iteratively improving V until it converges to V^*.

The implementation of the algorithm in this project uses the idea of Q-values, as given by the pseudocode below:

Algorithm 1 Value Iteration Algorithm

```
Require: $\pi$, the policy to be evaluated
  Initialize $V(s) = 0$, for all $s \in \mathcal{S}$
  repeat
    $\Delta \leftarrow 0$
    for each $s \in \mathcal{S}$ do
      for each $a \in \mathcal{A}$ do
        $Q(s,a) \leftarrow R(s) + \sum_{s' \in S} \gamma P_a(s'|s)V(s')$
      end for
      $\Delta \leftarrow \max(\Delta, |\max_{a \in \mathcal{A}} Q(s,a) - V(s)|)$
      $V(s) \leftarrow \max_{a \in \mathcal{A}} Q(s,a)$
    end for
  until $\Delta < \theta$ (a small positive number)
Ensure: $V \approx V^*$
```

Value iteration converges asymptotically to the optimal policy as iterations continue: $V \to V^*$ as $i \to \infty$, where i is the number of iterations. However, in practice we terminate the algorithm when the largest change in values between iterations Δ reaches some pre-determined threshold θ.

2.1.2 Plot of Optimal Policy

For the given map and $\theta = 0.001$, the algorithm terminates at iteration **674**. Figure 2 shows the optimum policy discovered by this algorithm.

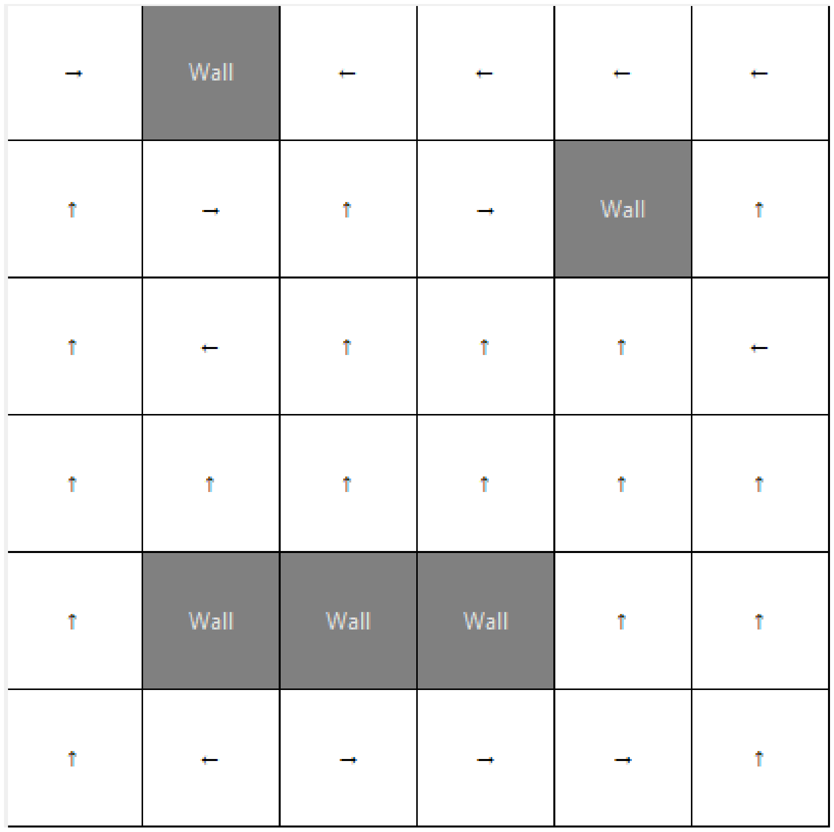
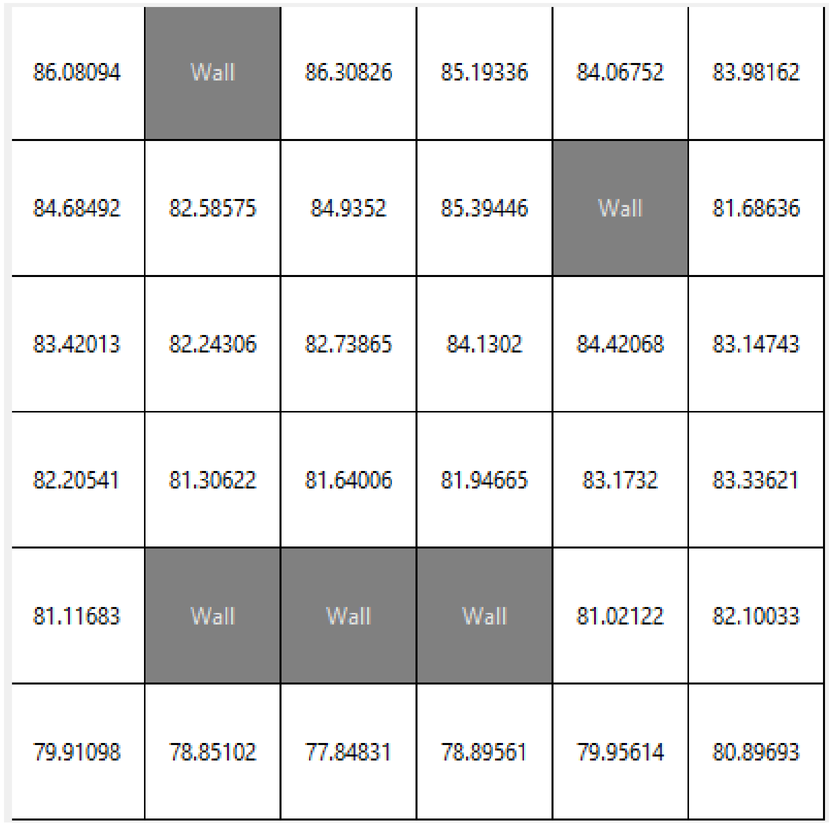


Figure 2: Optimum Policy found by Value Iteration

There are several interesting takeaways from the optimum policy. When the agent is in a state with positive rewards (+1.0) and there is an adjacent wall, the agent will opt to move into the wall to maximize its chances of staying still and collecting more rewards. The action taken by the agent in a state with negative rewards is also the shortest path to the state with positive rewards.

2.1.3 Utilities of States

Figure 3 shows the utilities of each state $V(S)$ after the algorithm has terminated.



86.08094	Wall	86.30826	85.19336	84.06752	83.98162
84.68492	82.58575	84.9352	85.39446	Wall	81.68636
83.42013	82.24306	82.73865	84.1302	84.42068	83.14743
82.20541	81.30622	81.64006	81.94665	83.1732	83.33621
81.11683	Wall	Wall	Wall	81.02122	82.10033
79.91098	78.85102	77.84831	78.89561	79.95614	80.89693

Figure 3: Estimated utilities of each state found by Value Iteration

As expected, the states with positive rewards typically have the highest utilities. However those states with positive rewards and an adjacent wall are assigned higher utilities, This is consistent with the observation of the optimum policy discussed in section 2.1.2, as the agent is able to collect more rewards in those states.

The states with negative rewards have their utilities proportional to the distance from the nearest state with positive rewards. This is again expected as the agent will need to take more moves before being rewarded positively.

2.1.4 Plot of estimated utilities

Figure 4 shows the plot of the estimated utilities as a function of the number of iterations. The estimated utilities $\sum_{s\in\mathcal{S}} V(s)$ approaches the optimum utility $\sum_{s\in\mathcal{S}} V^*(s)$ asymptotically as the number of iterations approaches infinity. Thus, we can estimate the optimum utility to be around 2500.

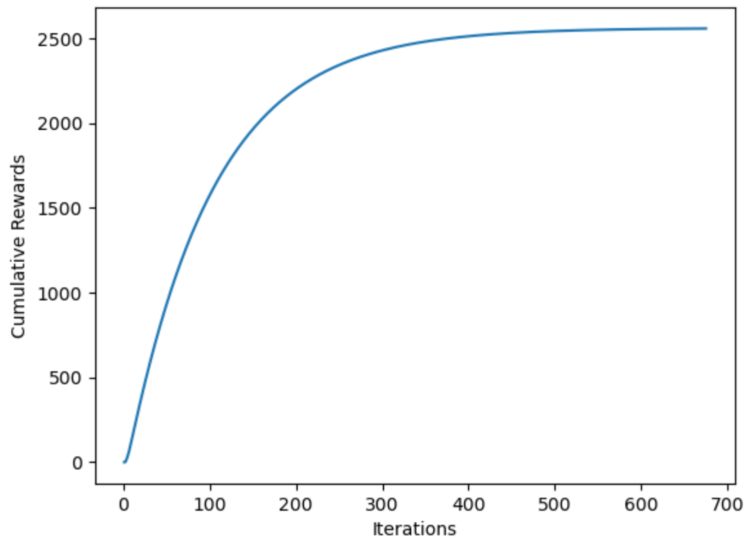


Figure 4: Plot of estimated utilities

2.2 Policy Iteration

2.2.1 Description

While Value Iteration iterates over values, Policy Iteration iterates over policies themselves, creating a strictly improved policy in each iteration (except if the iterated policy is already optimal). Policy Iteration first starts with some (non-optimal) policy, such as a random policy, and then calculates the value of each state of the MDP given that policy — this step is called policy evaluation. It then updates the policy itself for every state by calculating the expected reward of each action applicable from that state - this step is called policy evaluation.

The implementation of the algorithm used in this project is given by the pseudocode below:

Algorithm 2 Policy Evaluation Algorithm

```
Require: $\pi$, the policy to be evaluated
  repeat
    $\Delta \leftarrow 0$
    for each $s \in \mathcal{S}$ do
      $V'(s) \leftarrow R(s) + \sum_{s' \in S} \gamma P_a(s'|s)V(s')$
      $\Delta \leftarrow \max(\Delta, |V'(s) - V(s)|)$
    end for
    $V \leftarrow V'$
  until $\Delta < \theta$ (a small positive number)
```

Algorithm 3 Policy Iteration Algorithm

```
Require: $\pi$, the policy to be evaluated
  Initialize $V(s) = 0$, for all $s \in \mathcal{S}$
  Initialize $a =$ Left, for all $s \in \mathcal{S}$
  repeat
    Compute $V(s)$ for all $s$ using policy evaluation
    for each $s \in \mathcal{S}$ do
      $\pi(s) \leftarrow \text{argmax}_{a \in \mathcal{A}} R(s) + \sum_{s' \in S} \gamma P_a(s'|s)V(s')$
    end for
  until $\pi$ does not change
```

2.2.2 Plot of Optimal Policy

For the given map and $\theta = 0.001$, the algorithm terminates at iteration 5. Figure 5 shows the optimum policy discovered by this algorithm.

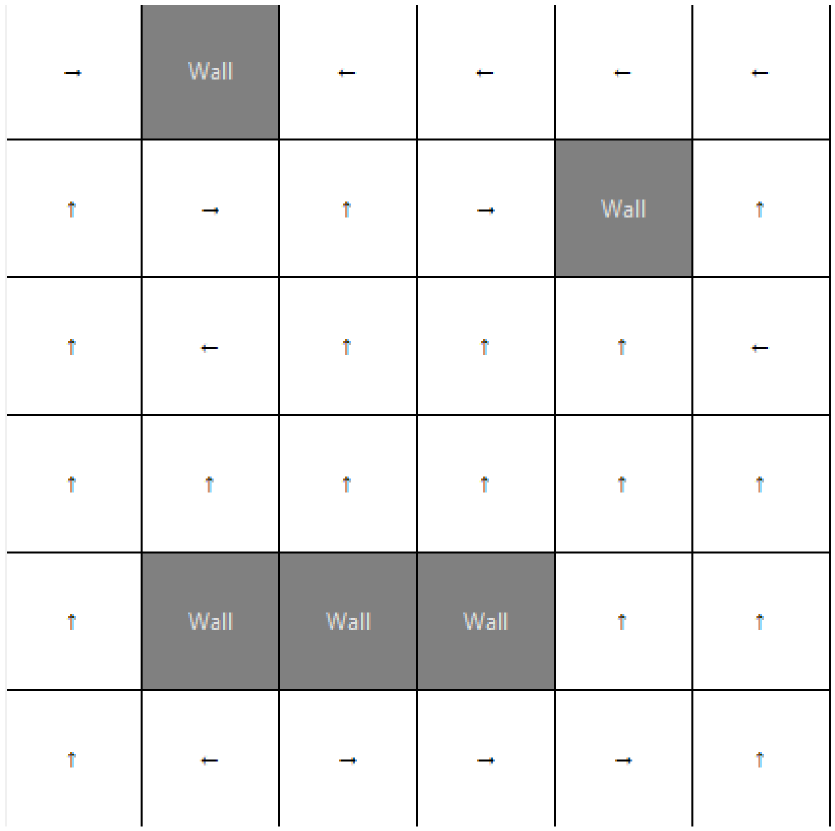
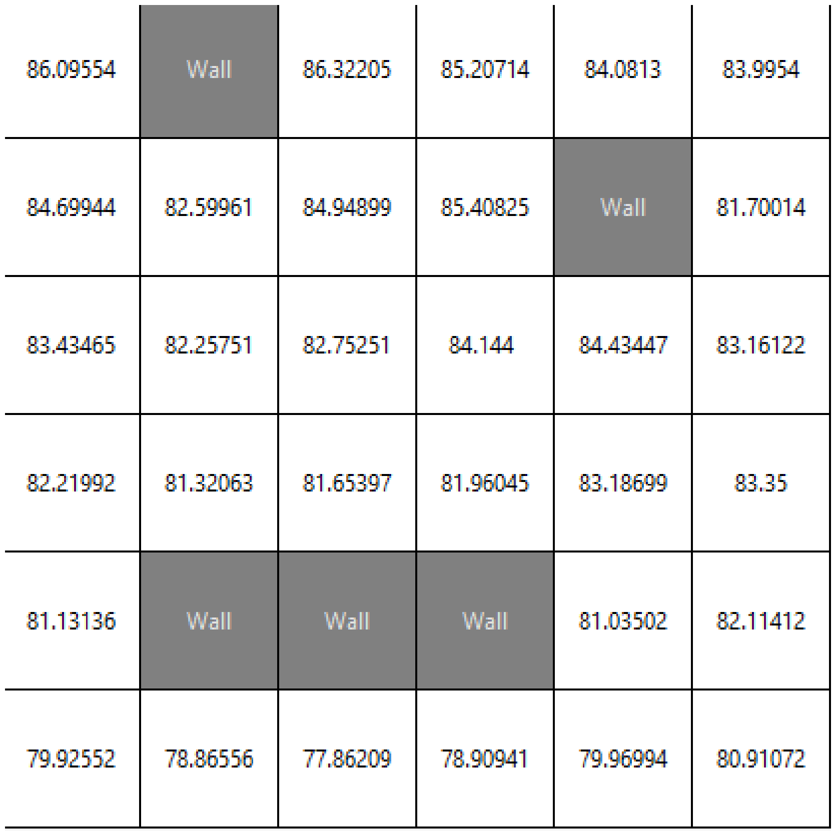


Figure 5: Optimum Policy found by Policy Iteration

The optimum policy found by Policy Iteration is the same as Value Iteration, which is to be expected as there is only one policy that is optimal. However, we note that policy iteration converges after much less iterations, since by fully evaluating the current policy in each iteration, the algorithm can make significant improvements in the policy.

2.2.3 Utilities of States

Figure 6 shows the utilities at each state $V(s)$ after the algorithm has terminated.



86.09554	Wall	86.32205	85.20714	84.0813	83.9954
84.69944	82.59961	84.94899	85.40825	Wall	81.70014
83.43465	82.25751	82.75251	84.144	84.43447	83.16122
82.21992	81.32063	81.65397	81.96045	83.18699	83.35
81.13136	Wall	Wall	Wall	81.03502	82.11412
79.92552	78.86556	77.86209	78.90941	79.96994	80.91072

Figure 6: Estimated utilities of each state found by Policy Iteration

The estimated utilities of each state found by Policy Iteration is similar to the estimated utilities of each state found by Value Iteration.

2.2.4 Plot of estimated utilities

Figure 7 shows the plot of the estimated utilities as a funciton of the number of iterations. The biggest increase in total estimated utilities was made during iteration 3, and the increases in the remaining iterations were small, suggesting the policy was close to optimum from iteration 3 onwards.

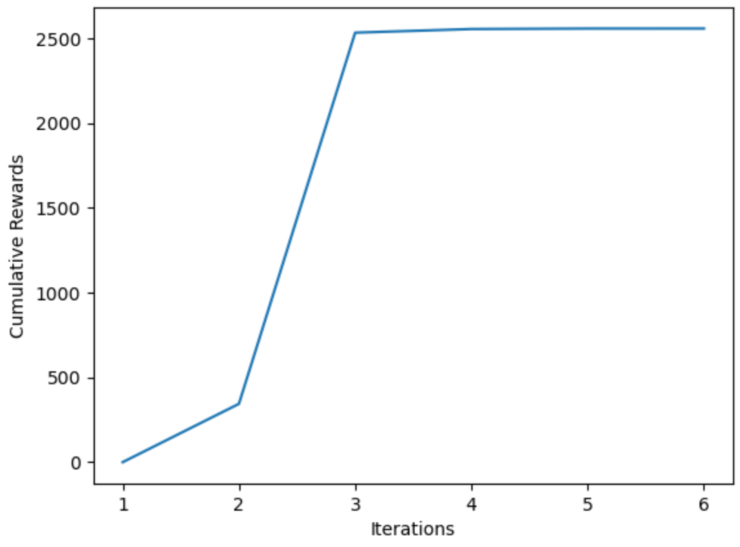


Figure 7: Plot of estimated utilities

3 Solving a more complex maze

In this section, we demonstrate the use case of Value and Policy Iteration in maze solving. Figure 8 shows a more complex maze, where the state having a reward of +1 representing the exit.

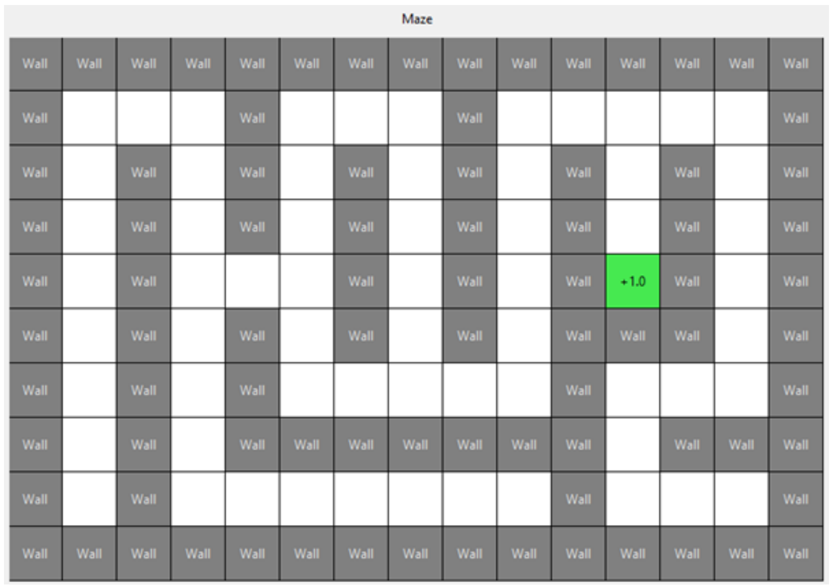


Figure 8: A more complex maze

We ran both Value and Policy Iteration on this maze and we observe the solution of the maze emerging after the algorithms have converged. The optimum policy found by the algorithms represent the shortest path to the maze exit.

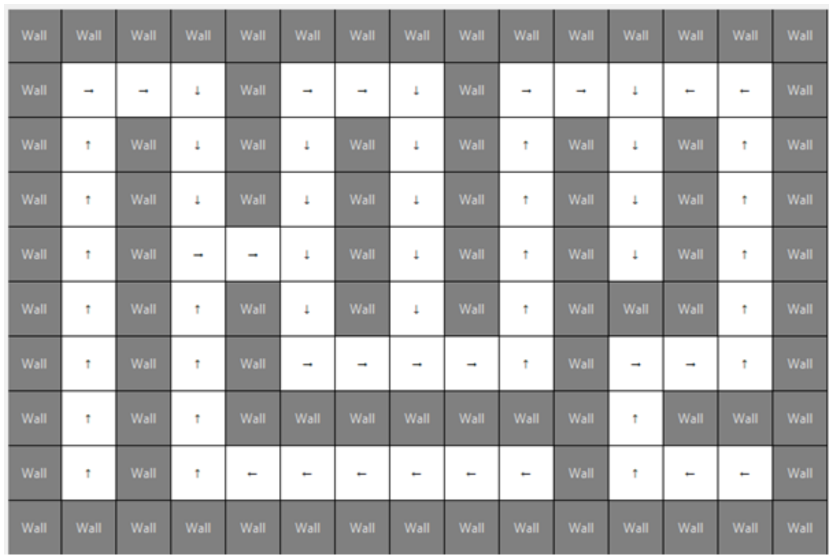


Figure 9: Solution to the more complex maze

4 Analysis of Algorithms

4.1 Time complexity of Value Iteration

First, we analyze the complexity of Value Iteration. On each iteration, we iterate in an outer loop over all states in S, and in each outer loop iteration, we need to iterate over all states ($\sum_{s \in S}$), which means $|S|^2$ iterations. Additionally, we need to calculate the value for every action to find the maximum. Hence, the complexity of each iteration is $O(|S|^2|A|)$. Additionally, the number of iterations are dependent on θ, and has no upper bound. Therefore we can conclude that Value Iteration works best with small-to-medium sized problems (where $|S|$ is manageable), and does not scale that well to larger sized problems.

4.2 Time complexity of Policy Iteration

Next, we analyze the complexity of Policy Iteration. As each state s has at most $|A|$ actions, Policy Iteration is guaranteed to terminate after at most $O(|A|^{|S|})$ iterations, unlike Value Iteration which can theoretically require infinite iterations. However, the time complexity of each iteration is $O(|S|^3 + |S|^2|A|)$, however in most situations Policy Iteration requires only a few iterations to converge. Again, we can conclude that Policy Iteration also works best with small-to-medium sized problems, and fails to scale well to larger sized problems.

4.3 Alternative Algorithms

For larger sized problems, we can adopt alternative algorithms such as Iterative / Modified Policy Iteration, where policy evaluation is done approximately. Such algorithms approximate the value function $V(s)$ by performing a limited number of updates (fewer than needed for exact convergence), which is much faster than the classical Value and Policy Iteration. Therefore, they scale better to larger sized problems as well.

5 Conclusion

This report covers both Value and Policy Iteration and demonstrate their effectiveness in solving MDPs. Such algorithms have multiple use cases and we showcase one use case: maze solving. Unfortunately, these classical algorithms scale poorly as the size of the environment increases, making them unsuitable for larger scale problems.