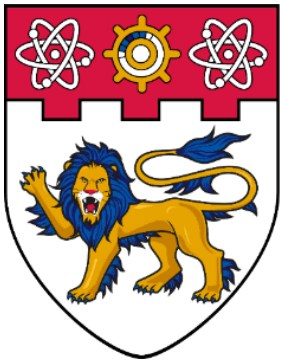


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

SC4003

INTELLIGENT AGENTS

Assignment 1: Agent Decision Making

Tan Choon Wee

U2120106H

COLLEGE OF COMPUTING AND DATA SCIENCE

NANYANG TECHNOLOGY UNIVERSITY

2025

Contents

1. Introduction

In this assignment, we explore the application of Markov Decision Processes (MDPs) to solve maze navigation problems using Value Iteration and Policy Iteration. These algorithms are fundamental to agent decision-making in artificial intelligence, enabling an agent to learn optimal policies in environments with stochastic transitions and rewards.

The maze environment is inspired by the example in Section 17.1 of the reference book "Artificial Intelligence: A Modern Approach" by Russell and Norvig. The transition model is stochastic: the agent moves in the intended direction with a probability of 0.8, and with a probability of 0.1, it moves at a right angle to the intended direction. If a move would result in hitting a wall, the agent remains in its current state.

Figure 1: Given maze environment and transition model

The rewards are defined as follows:

- White squares: -0.05
- Green squares: +1
- Brown squares: -1

The agent operates in an infinite-horizon environment with no terminal states, and a discount factor (γ) of 0.99 is used to ensure that future rewards are appropriately weighted.

This report documents the implementation of Value Iteration and Policy Iteration, the results obtained for both the given and custom maze environments, and an analysis of how environmental complexity impacts the performance of these algorithms. By the end of this assignment, we aim to demonstrate the effectiveness of MDP-based approaches in solving sequential decision-making problems and highlight the trade-offs between computational efficiency and solution quality.

2. Objectives

The report is structured into two main parts:

Part 1: Solving the Given Maze Environment

- Implement Value Iteration and Policy Iteration algorithms.
- Calculate and display the optimal policy and utilities of all non-wall states.
- Plot utility estimates as a function of the number of iterations, following the style of Figure 17.5(a) from the reference book.

Part 2: Designing and Solving a Complex Maze Environment

- Create a more intricate maze with a larger number of states and an increased proportion of wall states.
- Re-run Value Iteration and Policy Iteration algorithms on the complex maze.
- Analyse how the number of states and environmental complexity affect convergence.
- Explore the maximum complexity where the algorithms can still learn the right policy.

3. Implementation Overview

3.1 Tools and Libraries

- **Python 3.12.6**
  The programming language used for implementation.

- **Visual Studio Code (VSCode)**
  The integrated development environment (IDE) for writing and executing the code.

- **PyGame**
  Used for graphical representation of the maze environment, utilities, and policy results.

- **Jupyter Notebook**
  Used for the main execution file to provide an interactive and documented workflow.

3.2 Execution Instructions

To run the solution, simply execute the main file:

assignment_1_main.ipynb
This Jupyter Notebook serves as the main entry point for the assignment. Running this file will execute the entire solution, including the implementation of Value Iteration and Policy Iteration algorithms, visualization of results, and saving of output files.

3.3 File Structure

Root Directory Files

assignment_1_main.ipynb	The main execution file for the assignment.
config.py	Stores all predefined constants and variables, ensuring easy configuration and maintainability.
utils.py	Provides utility functions for file input/output operations, facilitating the saving and loading of results.

Table 1: Root Directory Files

The root directory also contains the following subdirectories:

algorithms

algorithm_utility.py	Contains functions for visualizing and displaying the maze environment, state utilities, and policies using PyGame.
policy_iteration.py	Implements the policy iteration algorithm.
value_iteration.py	Implements the value iteration algorithm.

asset

Stores images and resources used for visualization, such as arrow pictures for policy representation and markdown comments.

maze

Contains four maze configuration files, each defining a unique maze environment used for testing and comparing the algorithms.

Table 2: Subdirectories

3.4 Result folder

Upon successful execution of the main file, a "**result**" folder is automatically created to store the outputs of the policy and value iteration algorithms. The folder contains two sub-folders:

result

policy_iteration.py	Stores the policy iteration results for all four maze environments.
value_iteration.py	Stores the value iteration results for all four maze environments.

Table 3: Result folder

These results are saved and read during the execution of the main file for further analysis and comparison.

4. Using method of Value Iteration for Part 1

The given maze is visualised with configuration details:


```
# Maze size
MAZE_WIDTH  = 6
MAZE_HEIGHT = 6

# Starting position
START_POS = (2, 3)

# Green squares  – Count: 6
(0, 0), (2, 0), (5, 0), (3, 1), (4, 2),
(5, 3)

# Brown squares  – Count: 5
(1, 1), (5, 1), (2, 2), (3, 3), (4, 4)

# Wall locations – Count: 5
(2, 4), (3, 4), (1, 0), (4, 1), (1, 4)

# White squares  – Count: 20
```

Figure 2: Visualisation of given maze

4.1 Descriptions of Value Iteration implementation solutions

Value Iteration is a dynamic programming algorithm used to compute the optimal policy by iteratively updating the utility values of states until they converge. It is based on the Bellman equation, which expresses the utility of a state as the immediate reward plus the discounted utility of the next state.

The value iteration update rule from the basis of Bellman equation:

$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\, U_i(s')$$

Figure 3: Value iteration rule

where:

- $U_{i+1}(s)$: Updated utility of state s at iteration $i + 1$
- $R(s)$: Reward received in state s
- γ : Discount factor $(0 < \gamma \leq 1) \Rightarrow 0.99$ in this assignment
- $\max_{a \in A(s)}$: Maximization over all possible actions a available in state s
- $P(s' \mid s, a)$: Transition probability from state s to s' given action a

Value Iteration Method

- Start out with every $U(s) = 0$
- Iterate until convergence
  - During the *i*th iteration, update the utility of each state according to this rule:

$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\, U_i(s')$$

Figure 4: Value iteration rule

- In the limit of infinitely many iterations, guaranteed to find the correct utility values
  - In practice, don't need an infinite number of iterations…

function VALUE-ITERATION(mdp, ϵ) **returns** a utility function

inputs: mdp, an MDP with states S, actions $A(s)$, transition model *P(s' | s, a)*, rewards $R(s)$, discount γ

ϵ, the maximum error allowed in the utility of any state

local variables: U, U', vectors of utilities for states in S, initially zero

δ, the maximum change in the utility of any state in an iteration

repeat

$U \leftarrow U'; \delta \leftarrow 0$

for each state s **in** S **do**

$$U'(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\, U_i(s')$$

if $|U'[s] - U[s]| > \delta$ **then** $\delta \leftarrow |U'[s] - U[s]|$

until $\delta = \epsilon(1-\gamma)/\gamma$

return U

Figure 5: The value iteration algorithm for calculating utilities of states

4.2 Plot of optimal Value Iteration policy

```
Value iteration converged in 1145 iterations
Time taken for convergence: 0.7206 seconds
```

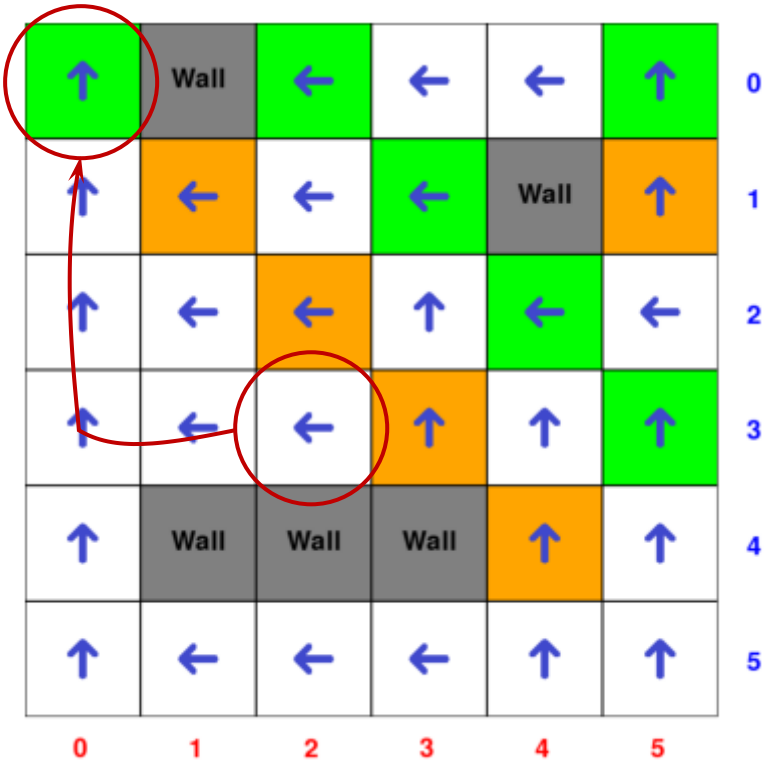


Figure 6: Plot of optimal Value Iteration policy

Optimal Value Iteration Policy Observation

- From the visualization of the optimal value iteration policy, we notice a clear tendency for the agent to navigate towards the top left corner.

4.3 Value Iteration utilities of all states

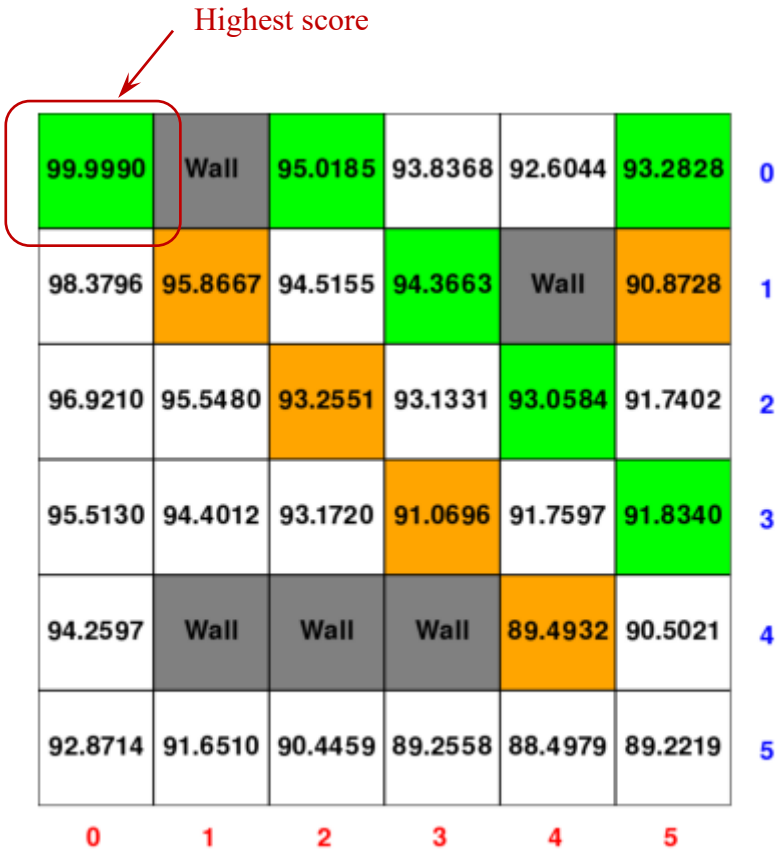


Figure 7: Value iteration utilities of all states

4.4 Plot of Value Iteration utility estimates as a function of the number of iterations (wall omitted)

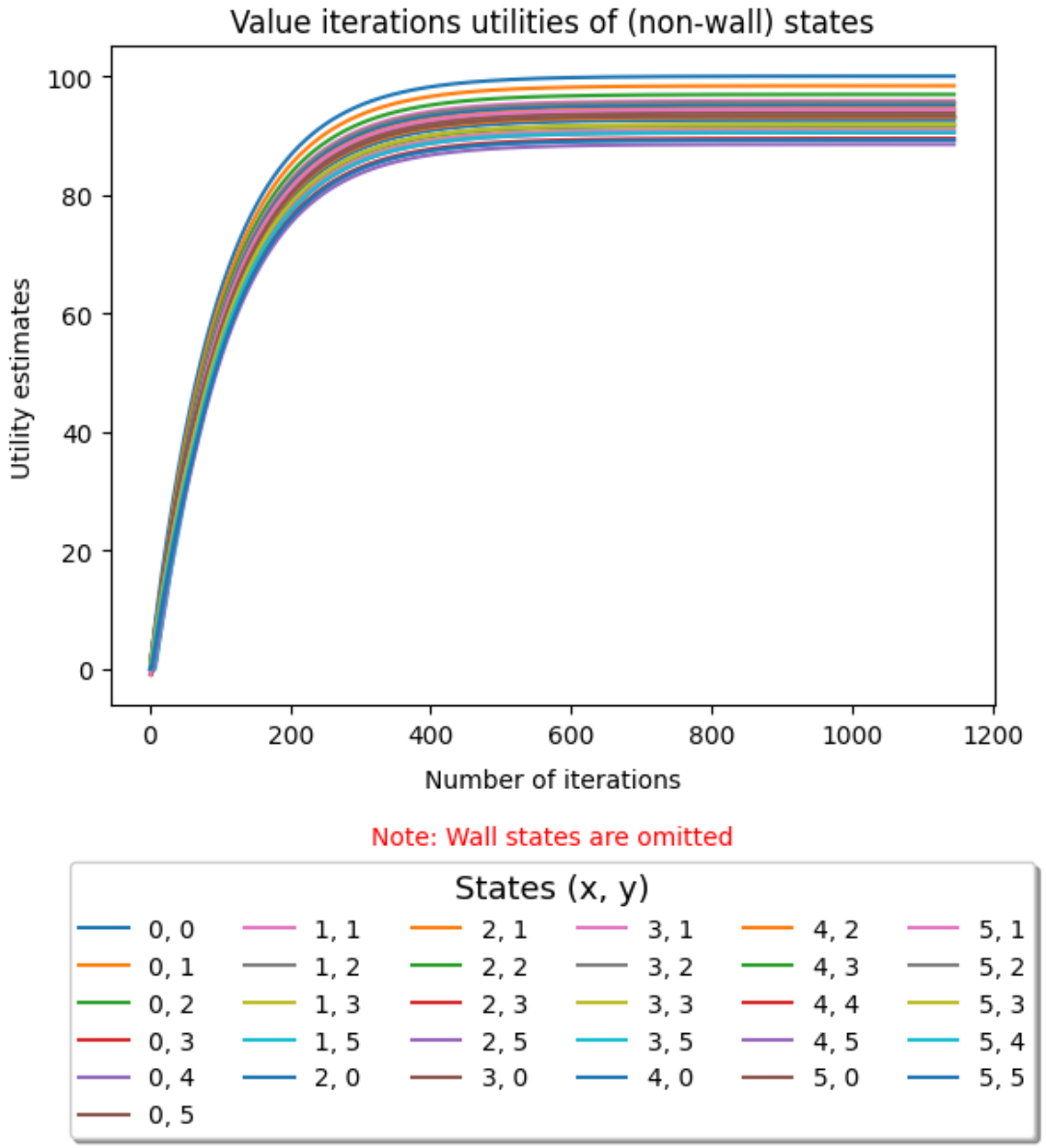


Figure 8: Plot of Value Iteration utility estimates as a function of the number of iterations

5. Using method of Policy Iteration for Part 1

5.1 Descriptions of Policy Iteration implementation solutions

Policy iteration is an iterative algorithm used to find the optimal policy in a Markov Decision Process (MDP). It alternates between policy evaluation and policy improvement until the policy stabilizes.

Policy Iteration Method

- Start with some initial policy π_0 and alternate between the following steps:
  - **Policy evaluation** : calculate $U^{\pi_i}(s)$ for every state s
  - **Policy improvement** : calculate a new policy π_{i+1} based on the updated utilities

$$\pi^{i+1}(s) = \arg\max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\, U^{\pi_i}(s')$$

Figure 9: Policy improvement equation

Policy Evaluation

- Given a fixed policy π, calculate $U^{\pi}(s)$ for every state s
- The Bellman equation for the optimal policy:

$$U(s) = R(s) + \max_{a} \sum_{s'} P(s' \mid s, a)\, U(s')$$

Figure 10: Bellman equation for optimal policy

- How does it need to change if our policy is fixed?

$$U^{\pi}(s) = R(s) + \gamma \sum_{s'} P\big((s' \mid s, \pi(s)\big)\, U^{\pi}(s')$$

Figure 11: Equation that solve linear system

- Can solve a linear system to get all the utilities!
- Alternatively, can apply the following update:

$$U_{i+1}(s) \leftarrow R(s) + \gamma \sum_{s'} P\big((s' \mid s, \pi_i(s)\big)\, U_i(s')$$

Figure 12: Updated policy iteration equation

function POLICY-ITERATION(mdp, ϵ) **returns** a policy

inputs: mdp, an MDP with states S, actions $A(s)$, transition model *P(s' | s, a)*,

local variables: U, a vector of utilities for states in S, initially zero

π, a policy vector indexed by state, initially random

repeat

$U \leftarrow \text{POLICY-EVALUATION}(\pi, U, mdp)$

$unchanged? \leftarrow \text{true}$

for each state s **in** S **do**

if $\max_{a \in A(s)} \sum_{s'} P(s'|s,a)U[s'] > \sum_{s'} P(s'|s,\pi[s])U[s']$ **then do**

$\pi[s] \leftarrow \underset{a \in A(s)}{\text{argmax}} \sum_{s'} P(s'|s,a)U[s']$

$unchanged? \leftarrow \text{false}$

until $\delta unchanged?$

return π

Figure 13: The policy iteration algorithm for calculating an optimal policy

5.2 Plot of optimal Policy Iteration policy

(Using seed number for reproducible results)

```
Value iteration converged in 6 iterations
Time taken for convergence: 0.1769 seconds
```

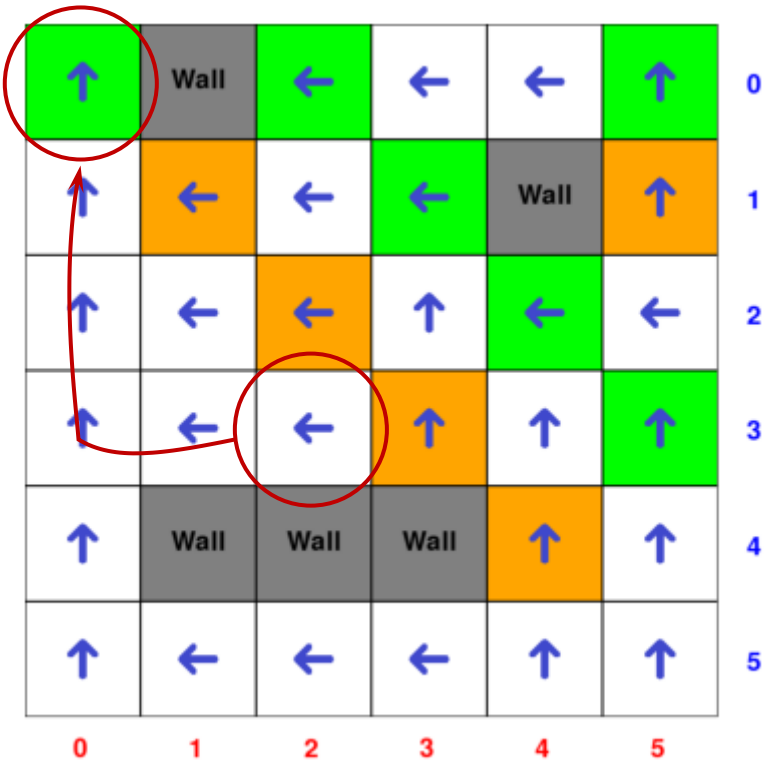


Figure 14: Plot of optimal Policy Iteration policy

5.3 Policy Iteration utilities of all states

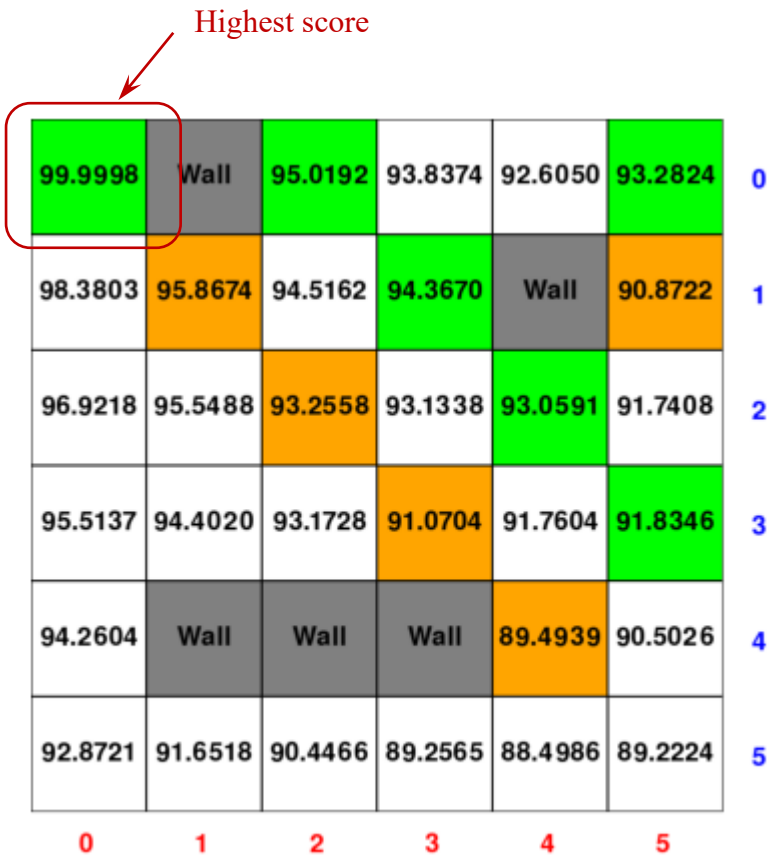


Figure 15: Policy iteration utilities of all states

5.4 Plot of Policy Iteration utility estimates as a function of the number of iterations (wall omitted)

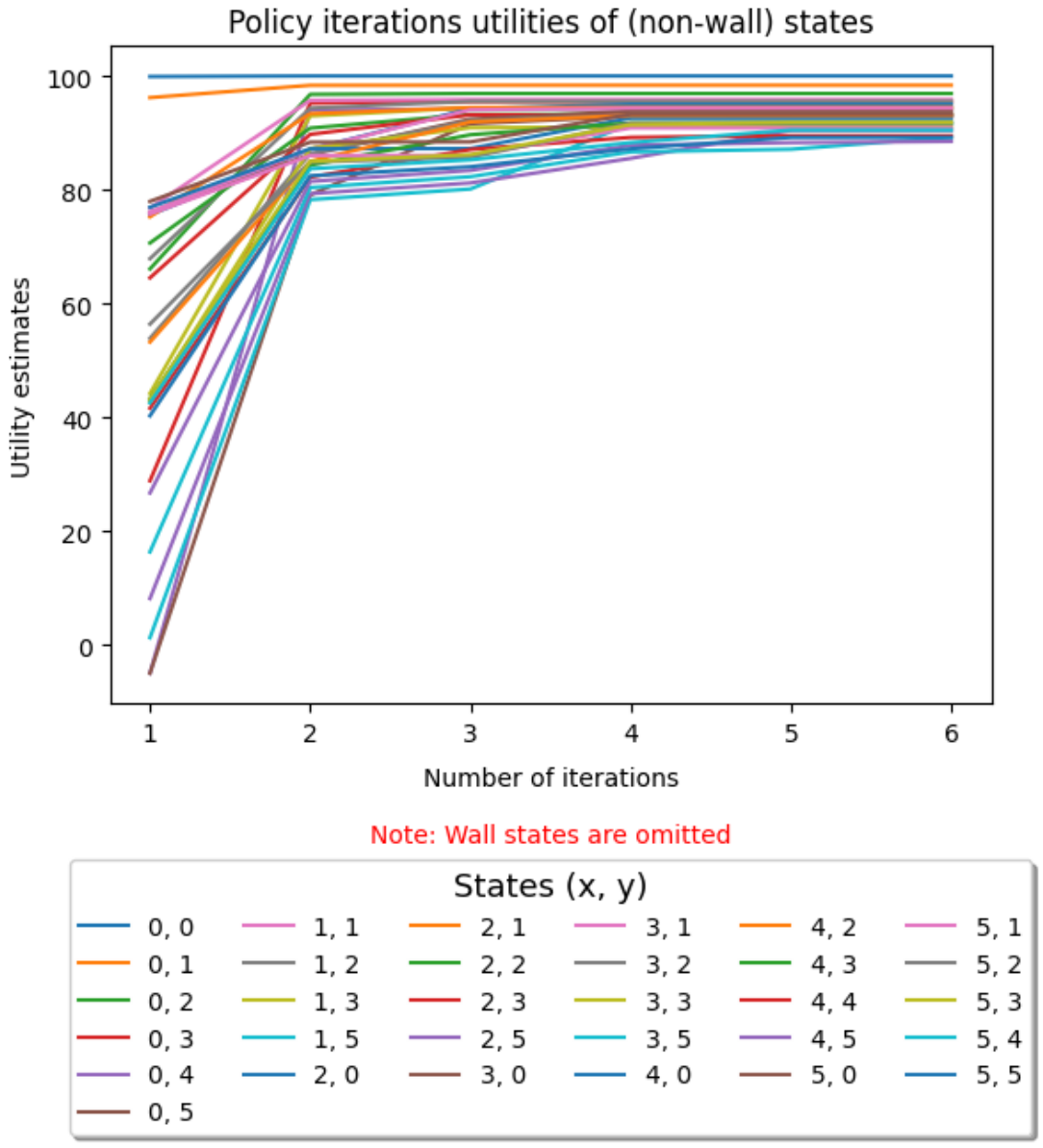


Figure 16: Plot of Policy Iteration utility estimates as a function of the number of iterations

Policy Iteration Observations

```
Given maze algorithm running results:
- There are 25 states in the maze

- Value iteration converged in 1145 iterations
- Time taken for convergence: 0.7206 seconds

- Policy iteration converged in 6 iterations
- Time taken for convergence: 0.1769 seconds

Amplitude of policy_iteration_time_1 over value_iteration_time_1 is
0.25
```

Part 1 Observation

- The visualization of the optimal Policy Iteration policy reveals that it is identical to the Value Iteration policy, confirming consistency between the two methods.

- Policy Iteration achieved significantly faster convergence compared to Value Iteration.

- Policy iteration demonstrates a shorter convergence time relative to value iteration.

6. More complicated maze environment

A 10x10 maze with random configurations is generated (using a seed number for reproducible results):

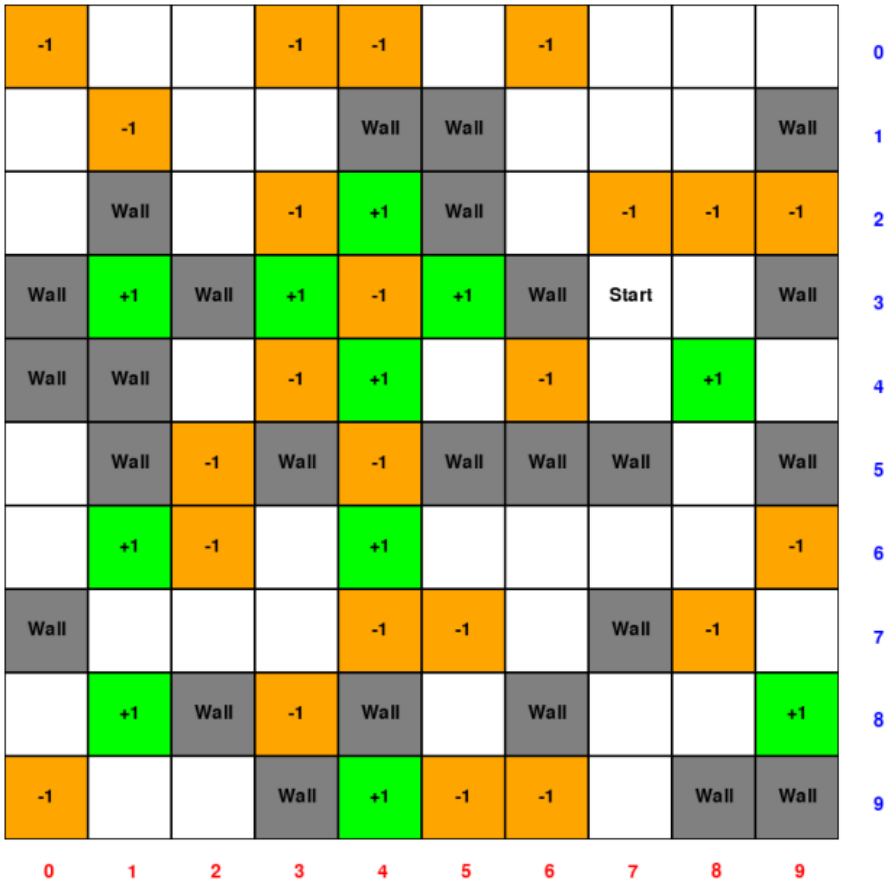


Figure 17: Visualisation of 10x10 maze

```
# Maze size
MAZE_WIDTH  = 10
MAZE_HEIGHT = 10

# Starting position
START_POS = (7, 3)

# Green squares  – Count: 11
(4, 4), (8, 4), (4, 9), (1, 8), (4, 6), (4, 2), (3, 3), (9, 8), (5, 3), (1, 6)
(1, 3)

# Brown squares  – Count: 23
(4, 0), (3, 4), (4, 3), (9, 2), (5, 7), (2, 5), (3, 0), (4, 5), (5, 9), (8, 2)
(6, 4), (3, 2), (4, 7), (3, 8), (0, 0), (8, 7), (1, 1), (9, 6), (0, 9), (2, 6)
(7, 2), (6, 0), (6, 9)

# Wall locations – Count: 25
(5, 1), (9, 5), (8, 9), (2, 8), (7, 7), (6, 5), (6, 8), (3, 9), (4, 8), (9, 1)
(0, 7), (1, 2), (0, 4), (1, 5), (4, 1), (3, 5), (5, 2), (5, 5), (9, 3), (9, 9)
(0, 3), (1, 4), (2, 3), (7, 5), (6, 3)

# White squares  – Count: 41
```

6.1 Run the Value Iteration Algorithm

```
Value iteration converged in 1145 iterations
Time taken for convergence: 1.2521 seconds
```

6.2 Plot of optimal Value Iteration policy

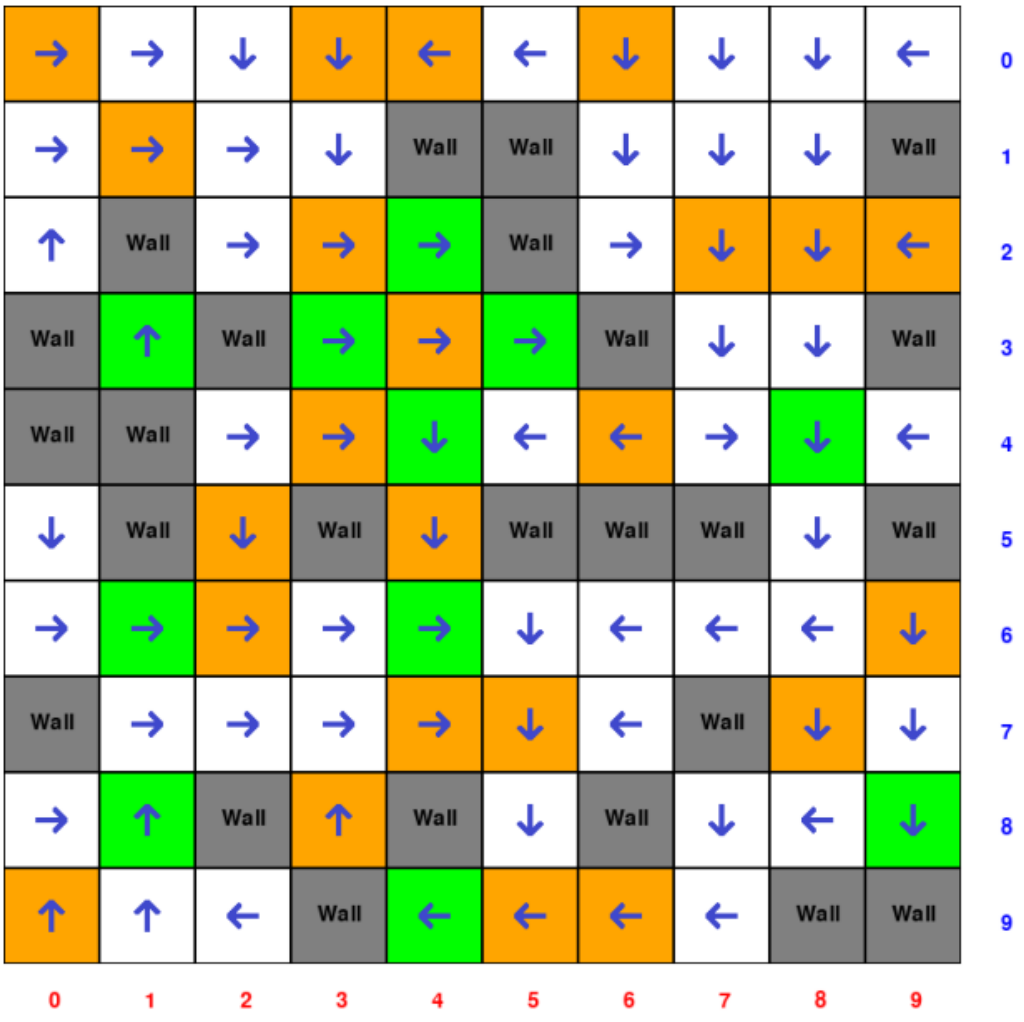


Figure 18: Plot of optimal Value Iteration policy

6.3 Value Iteration utilities of all states

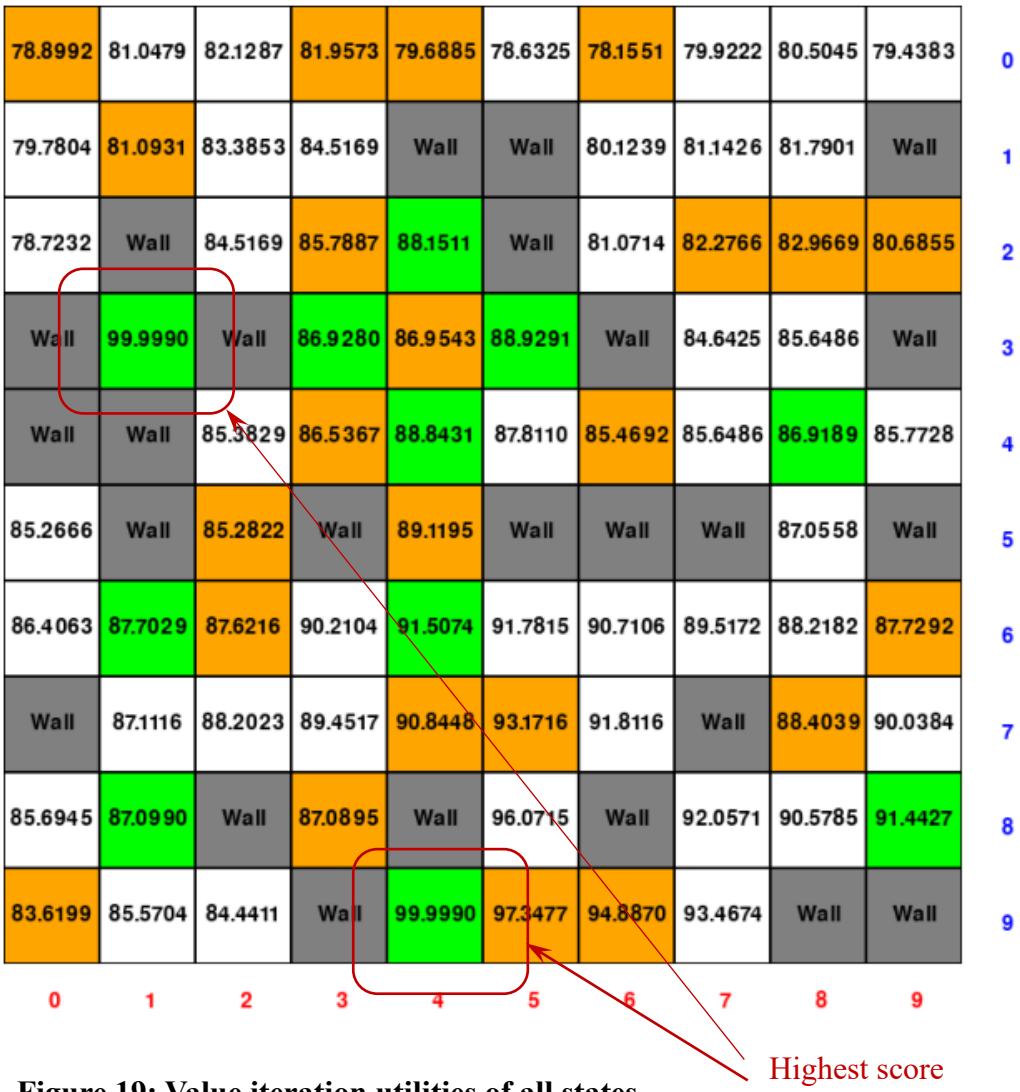


Figure 19: Value iteration utilities of all states

6.4 Plot of Value Iteration utility estimates as a function of the number of iterations (wall omitted)

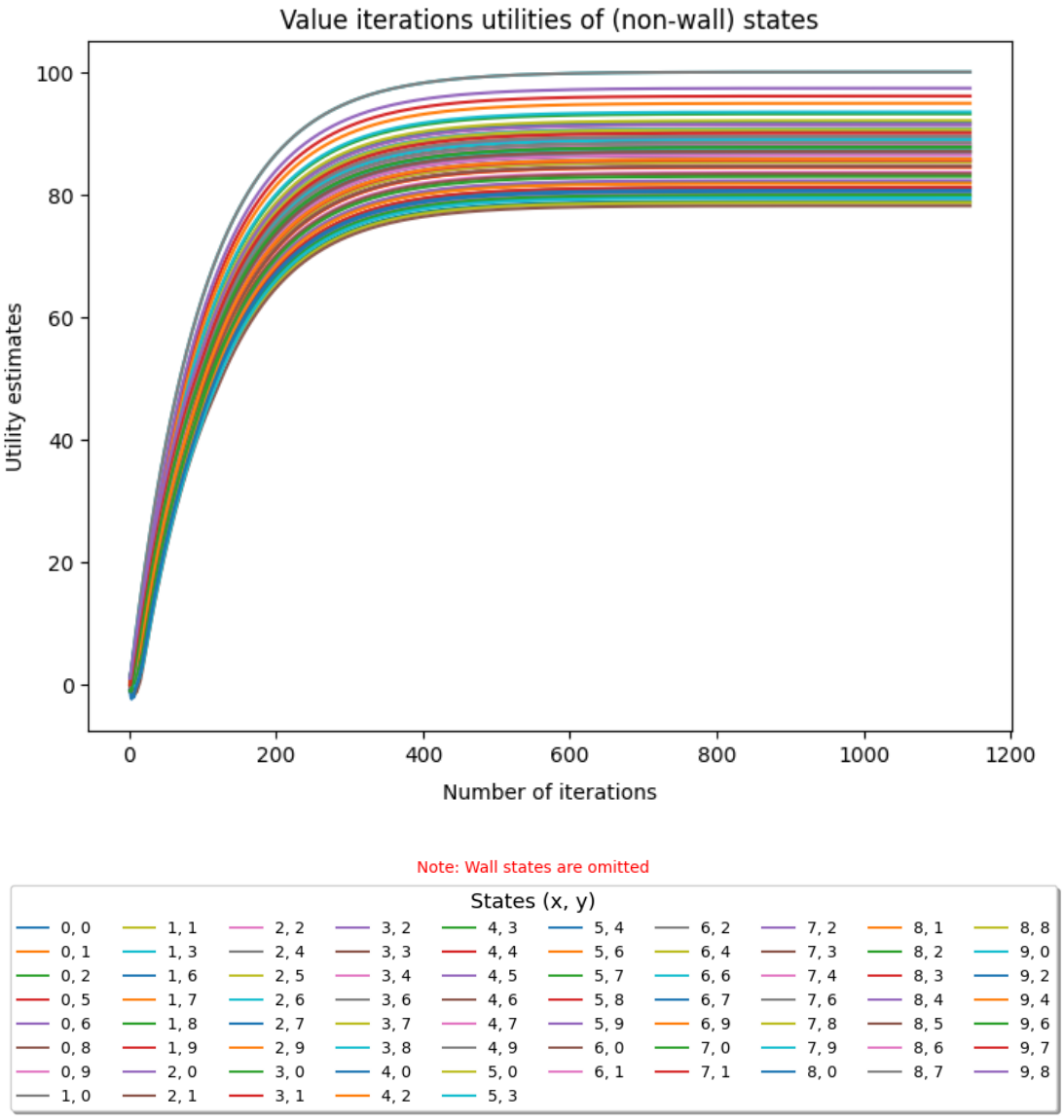


Figure 20: Plot of Value Iteration utility estimates as a function of the number of iterations

Value Iteration Observations

- Comparing given maze and 10x10 maze

```
Given maze algorithm running results:
- There are 25 states in the maze
- Value iteration converged in 1145 iterations
- Time taken for convergence: 0.7206 seconds

10x10 maze algorithm running results:
- There are 100 states in the maze
- Value iteration converged in 1145 iterations
- Time taken for convergence: 1.2521 seconds

Amplitude of 10x10_time_2 over given_time_1 is 1.74
```

- Although the 10x10 maze contains more states than the given maze, they both achieve convergence in an equal number of iterations.

- The convergence time for the 10x10 maze is twice longer than that of the given maze.

7.1 Run the Policy Iteration Algorithm

(Using seed number for reproducible results)

```
Policy iteration converged in 9 iterations
Time taken for convergence: 1.0036 seconds
```

7.2 Plot of optimal Policy Iteration policy

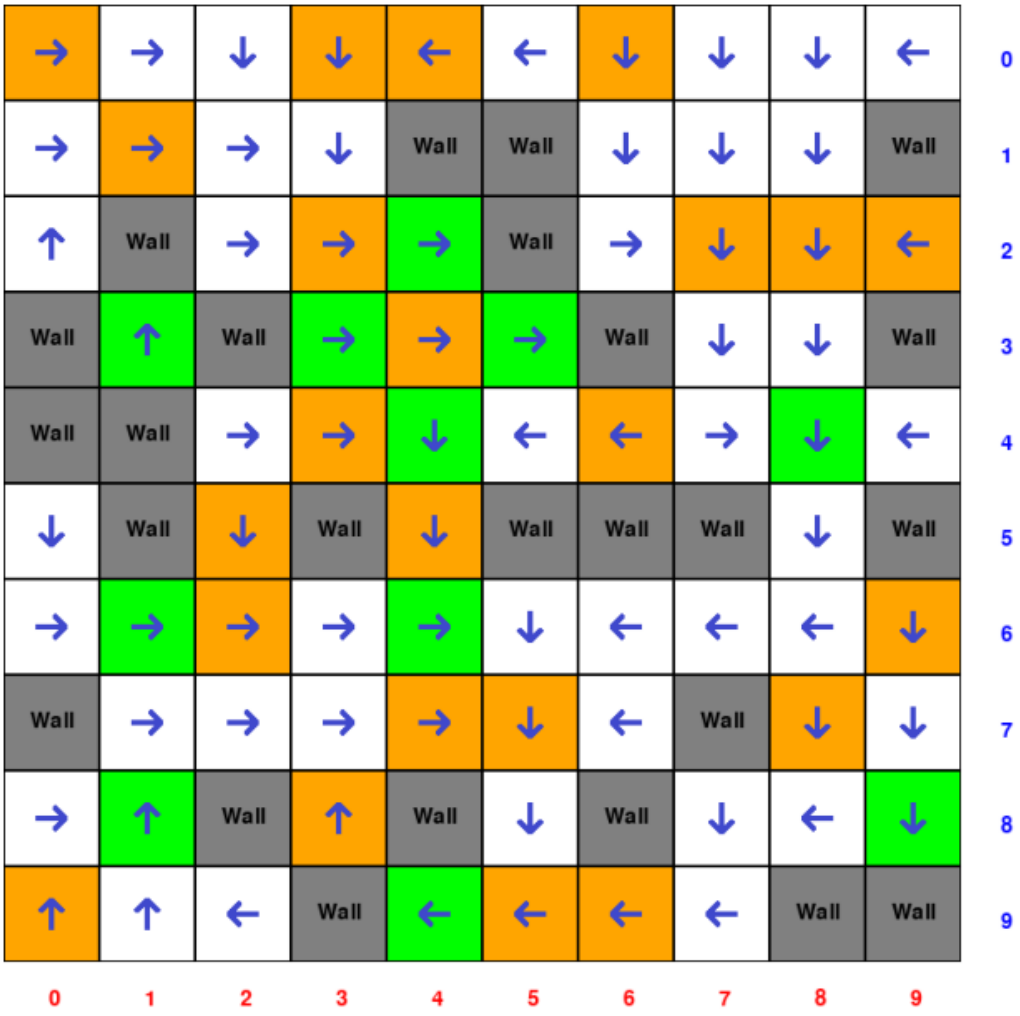


Figure 21: Plot of optimal Policy Iteration policy

7.3 Policy Iteration utilities of all states

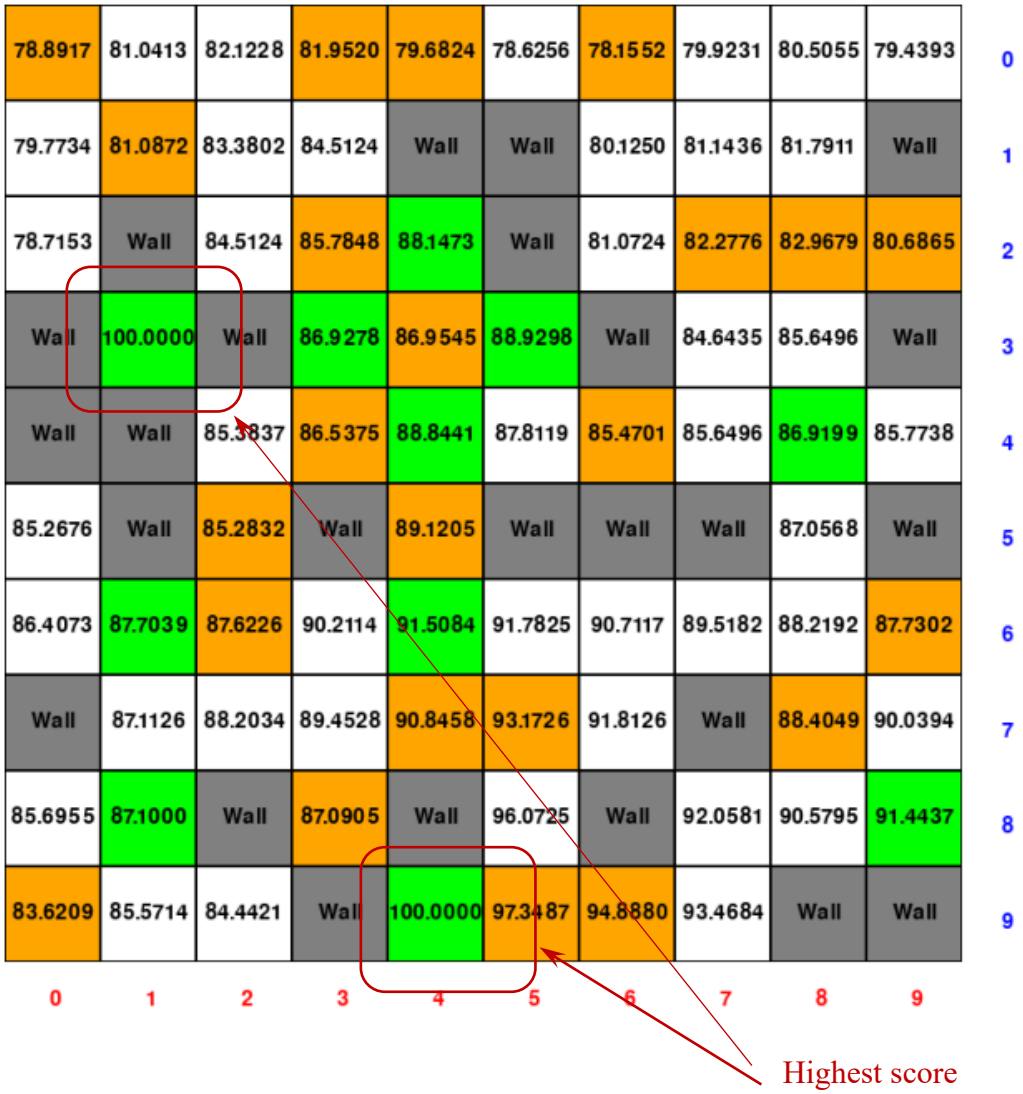


Figure 22: Policy iteration utilities of all states

7.4 Plot of Policy Iteration utility estimates as a function of the number of iterations (wall omitted)

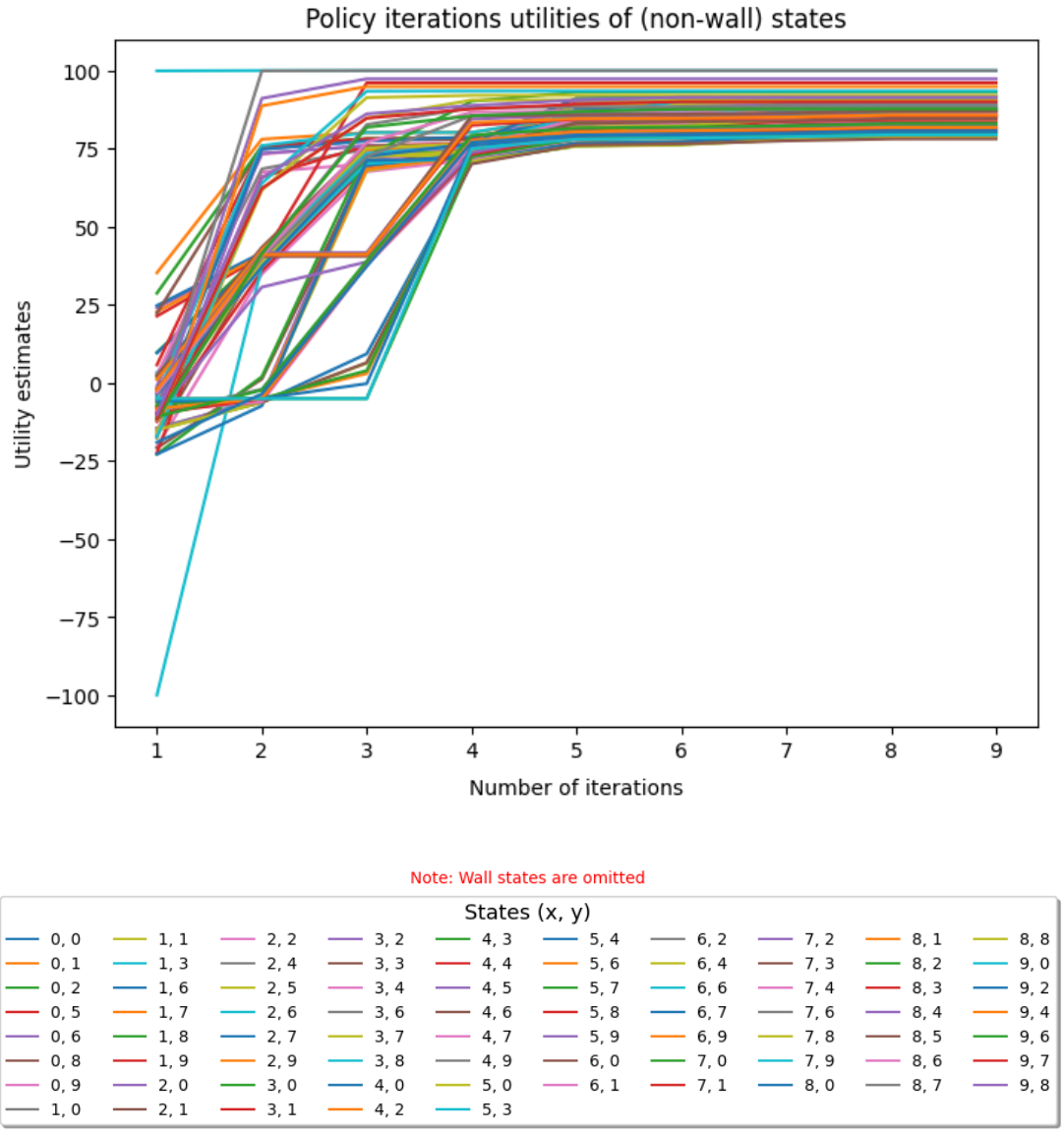


Figure 23: Plot of Policy Iteration utility estimates as a function of the number of iterations

Policy Iteration Observations

- Comparing given maze and 10x10 maze

```
Given maze algorithm running results:
- There are 25 states in the maze
- Policy iteration converged in 6 iterations
- Time taken for convergence: 0.1769 seconds

10x10 maze algorithm running results:
- There are 100 states in the maze
- Policy iteration converged in 9 iterations
- Time taken for convergence: 1.0036 seconds

Amplitude of 10x10_time_2 over given_time_1 is 5.67
```

- The number of iterations for convergence grows as the number of states increases.

- The convergence time for the 10x10 maze is over 5 times longer than that of the given maze.

8. Testing with complex maze environment

A 50x50 maze with random configurations is generated (using a seed number for reproducible results):

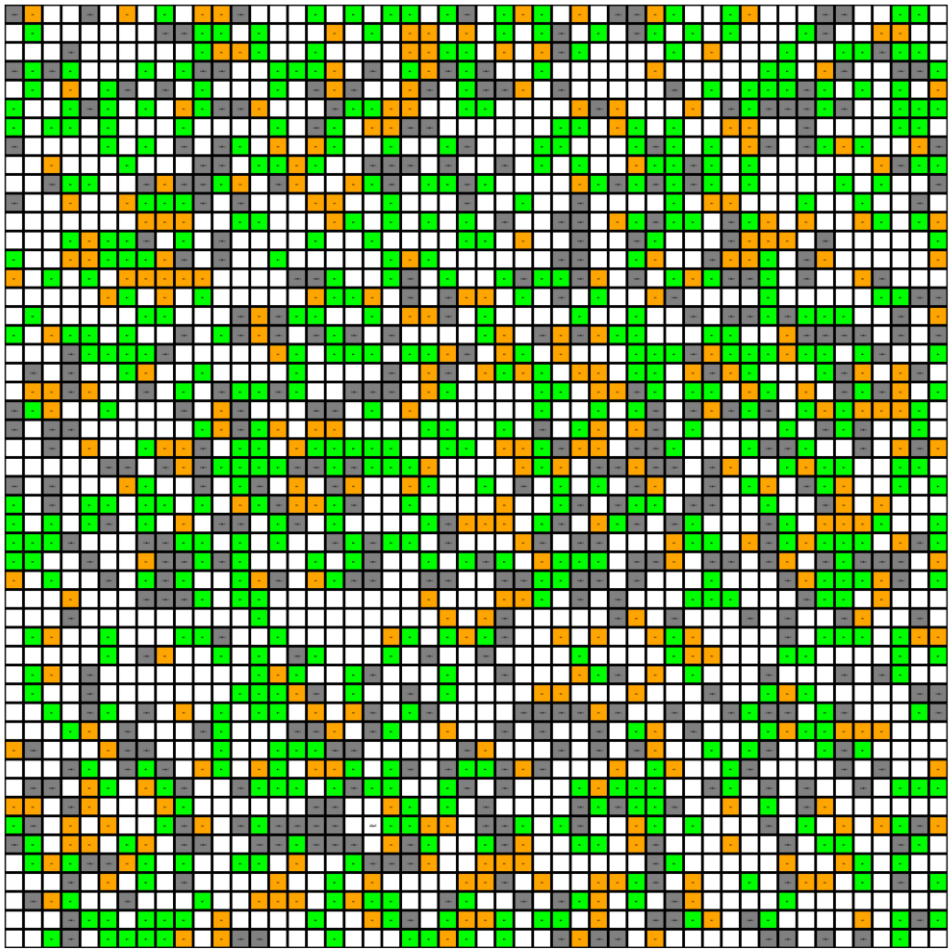


Figure 24: Visualisation of 50x50 maze

```
# Maze size
MAZE_WIDTH  = 50
MAZE_HEIGHT = 50

# Starting position
START_POS = (19, 43)

# Green squares  – Count: 559
# Brown squares  – Count: 322
# Wall locations – Count: 422
# White squares  – Count: 1197
```

8.1 Run the Value Iteration Algorithm

```
Value iteration converged in 1145 iterations
Time taken for convergence: 32.6772 seconds
```

8.2 Plot of optimal Value Iteration policy

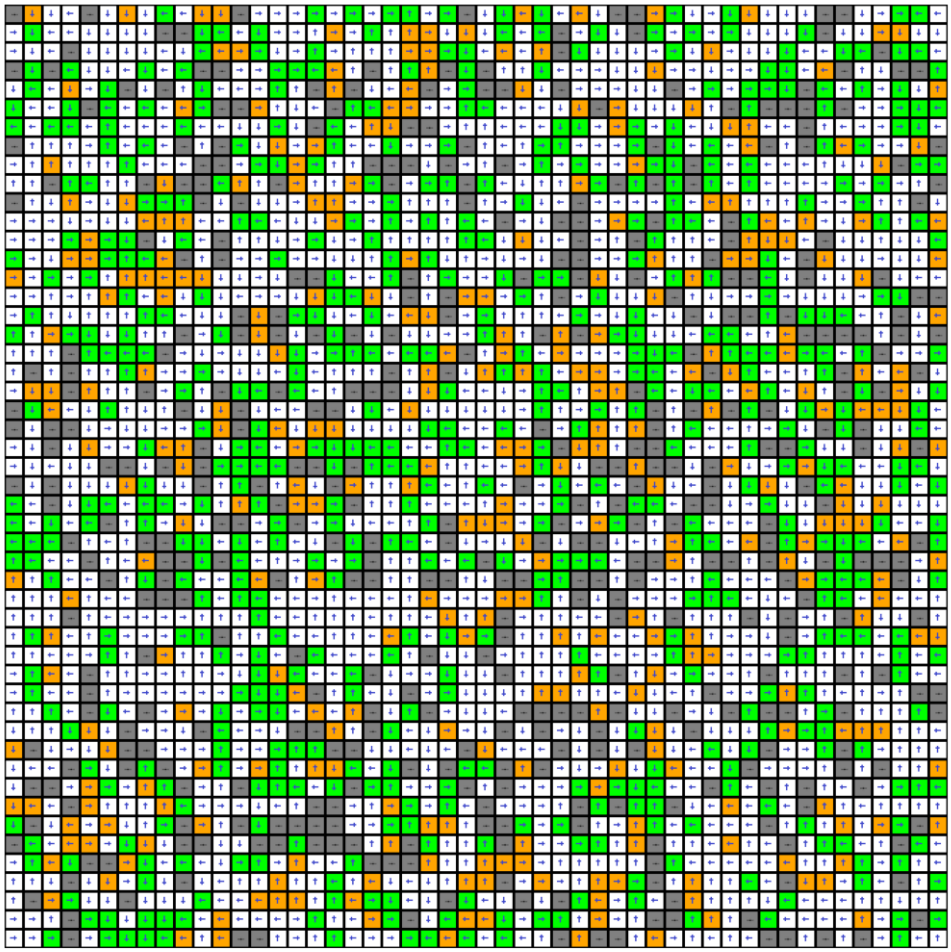


Figure 25: Plot of optimal Value Iteration policy

8.3 Value Iteration utilities of all states

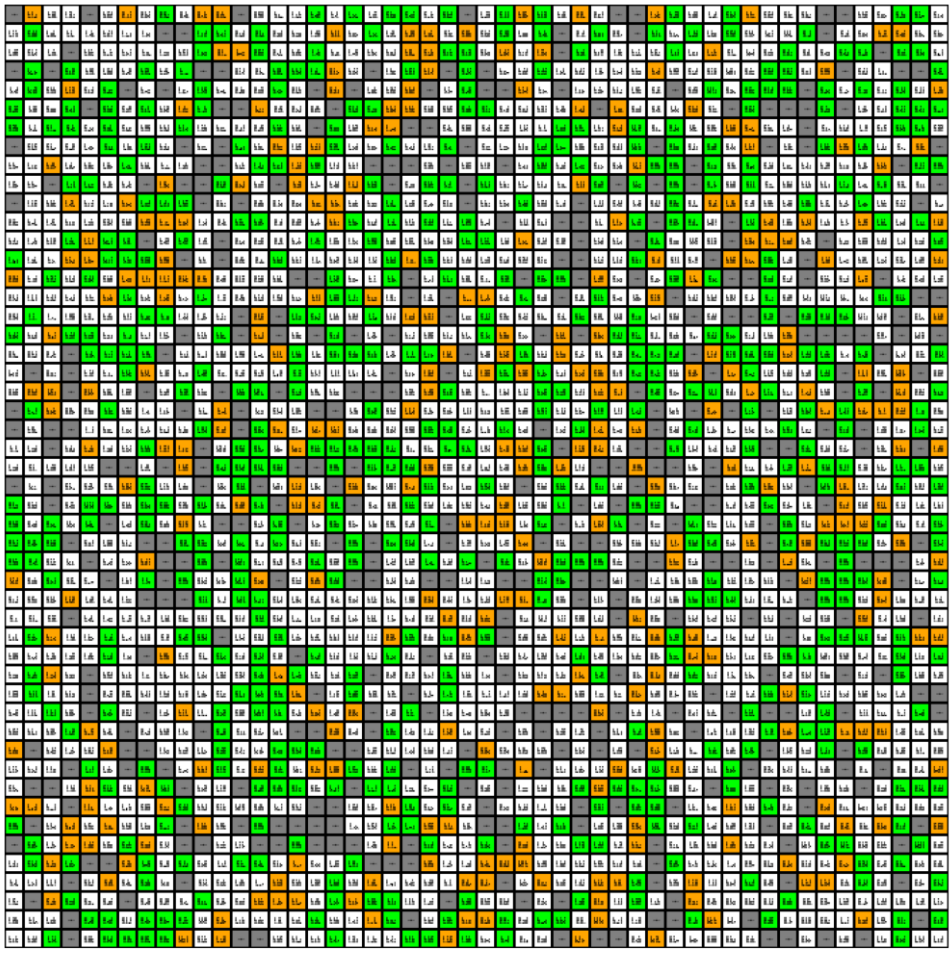
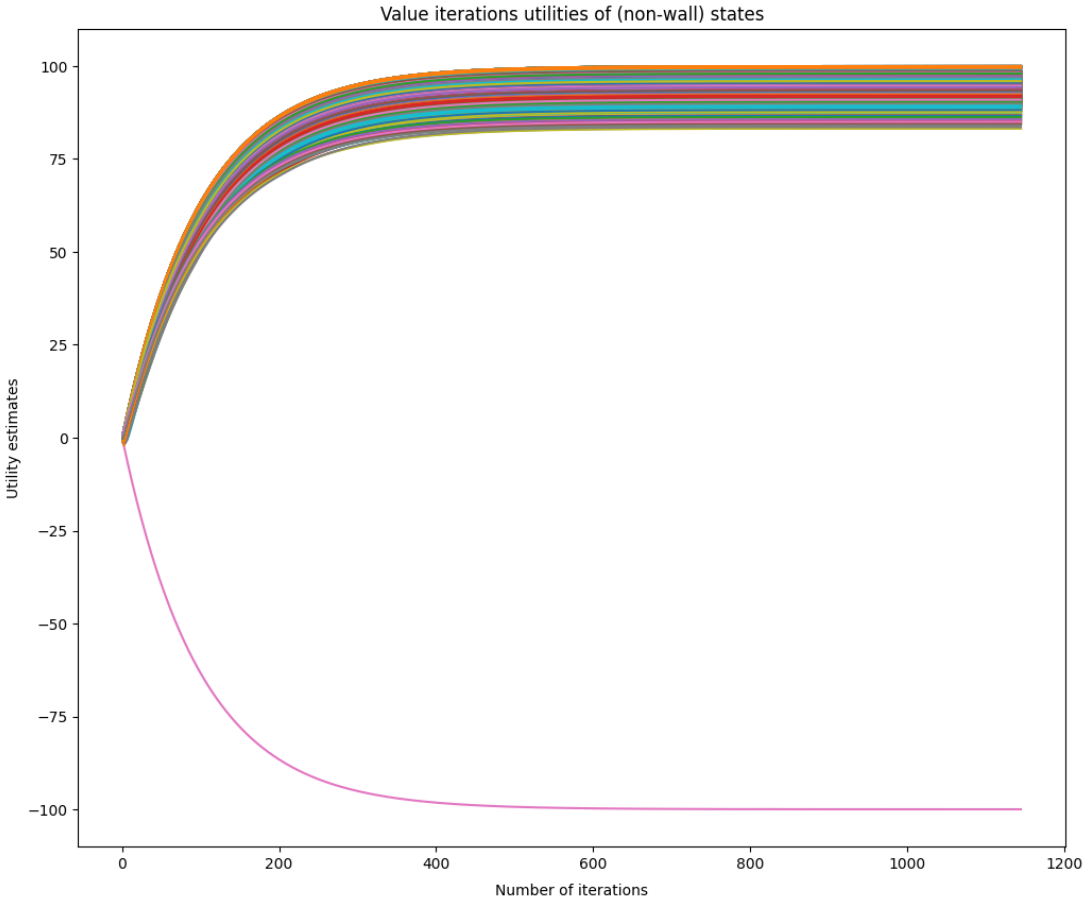


Figure 26: Value iteration utilities of all states

8.4 Plot of Value Iteration utility estimates as a function of the number of iterations (wall omitted)



Note: Wall states are omitted

Figure 27: Plot of Value Iteration utility estimates as a function of the number of iterations

Value Iteration Observations

- Comparing given maze, 10x10 maze, and 50x50 maze

```
Given maze algorithm running results:
- There are 25 states in the maze
- Value iteration converged in 1145 iterations
- Time taken for convergence: 0.7206 seconds

10x10 maze algorithm running results:
- There are 100 states in the maze
- Value iteration converged in 1145 iterations
- Time taken for convergence: 1.2521 seconds

50x50 maze algorithm running results:
- There are 2,500 states in the maze
- Value iteration converged in 1145 iterations
- Time taken for convergence: 32.6772 seconds

Amplitude of 10x10_time_2 over given_time_1 is 1.74
Amplitude of 50x50_time_3 over 10x10_time_2 is 26.10
Amplitude of 50x50_time_3 over given_time_1 is 45.35
```

- The number of iterations for convergence is identical across all three maze sizes.

- The convergence time for the 50x50 maze is over 20 times longer than that of the 10x10 maze.

- The convergence time for the 50x50 maze is over 40 times longer than that of the given maze.

9.1 Run the Policy Iteration Algorithm

(Using seed number for reproducible results)

```
Policy iteration converged in 12 iterations
Time taken for convergence: 31.4488 seconds
```

9.2 Plot of optimal Policy Iteration policy

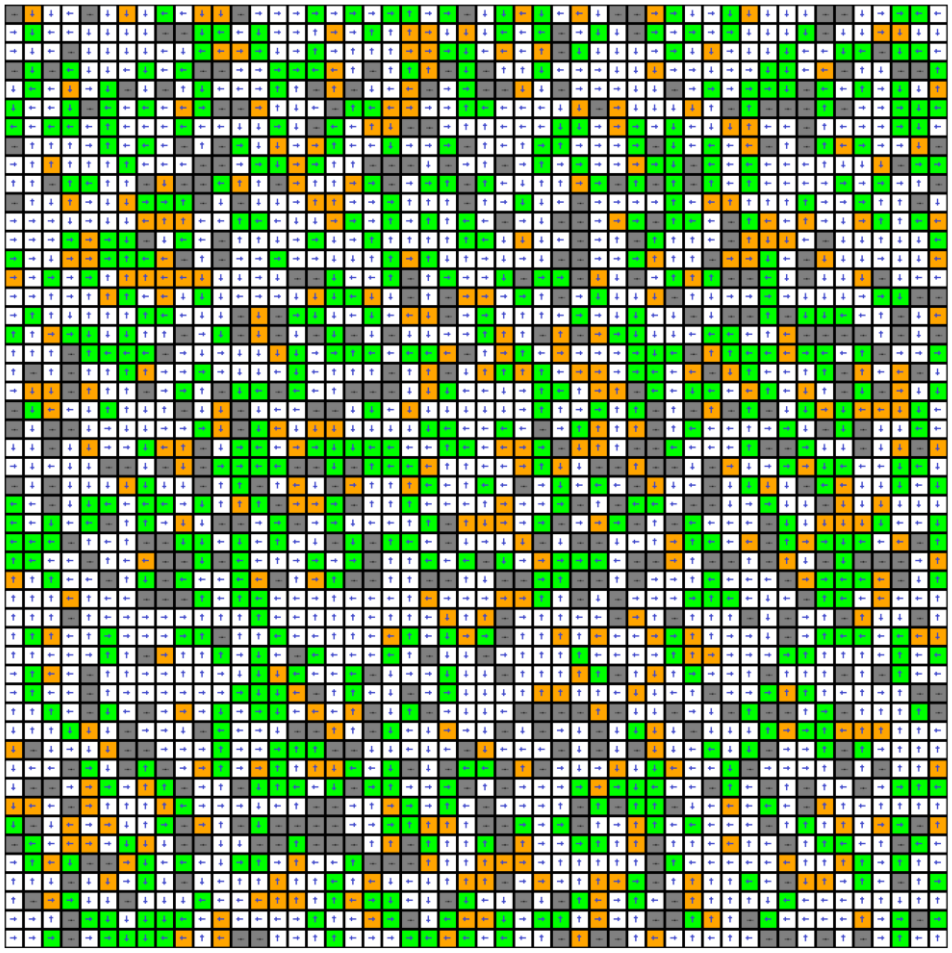


Figure 28: Plot of optimal Policy Iteration policy

9.3 Policy Iteration utilities of all states

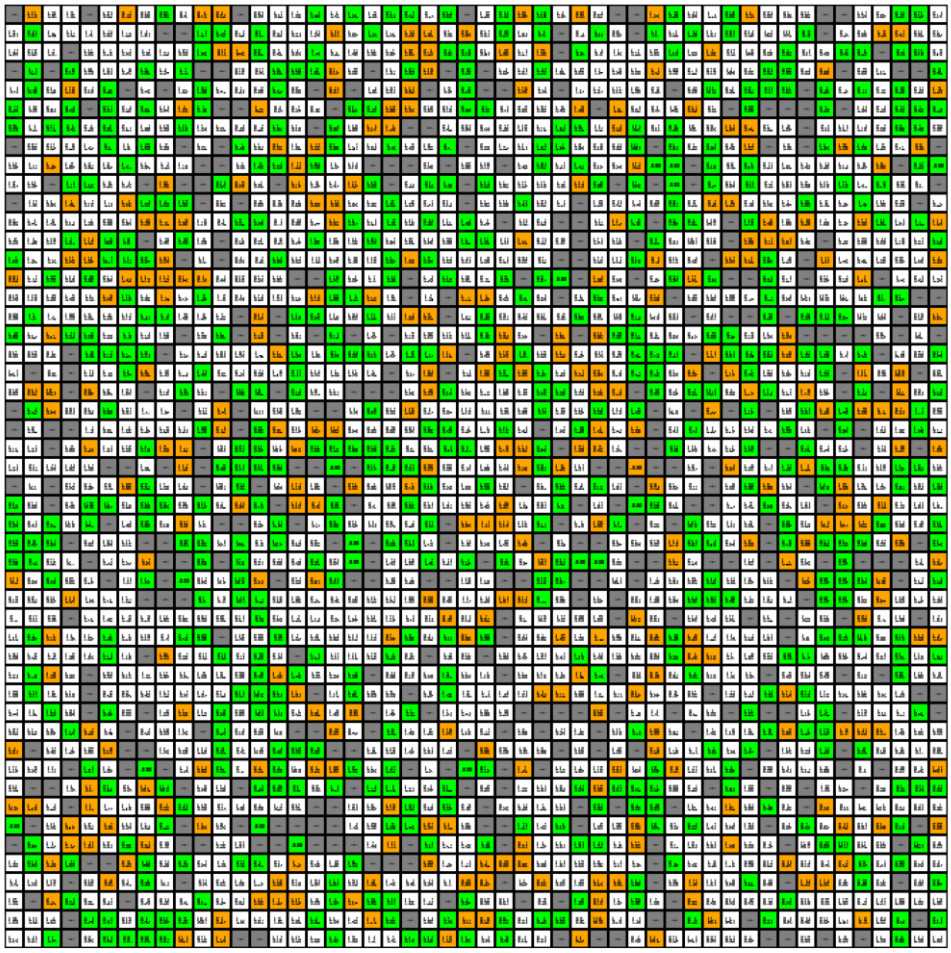
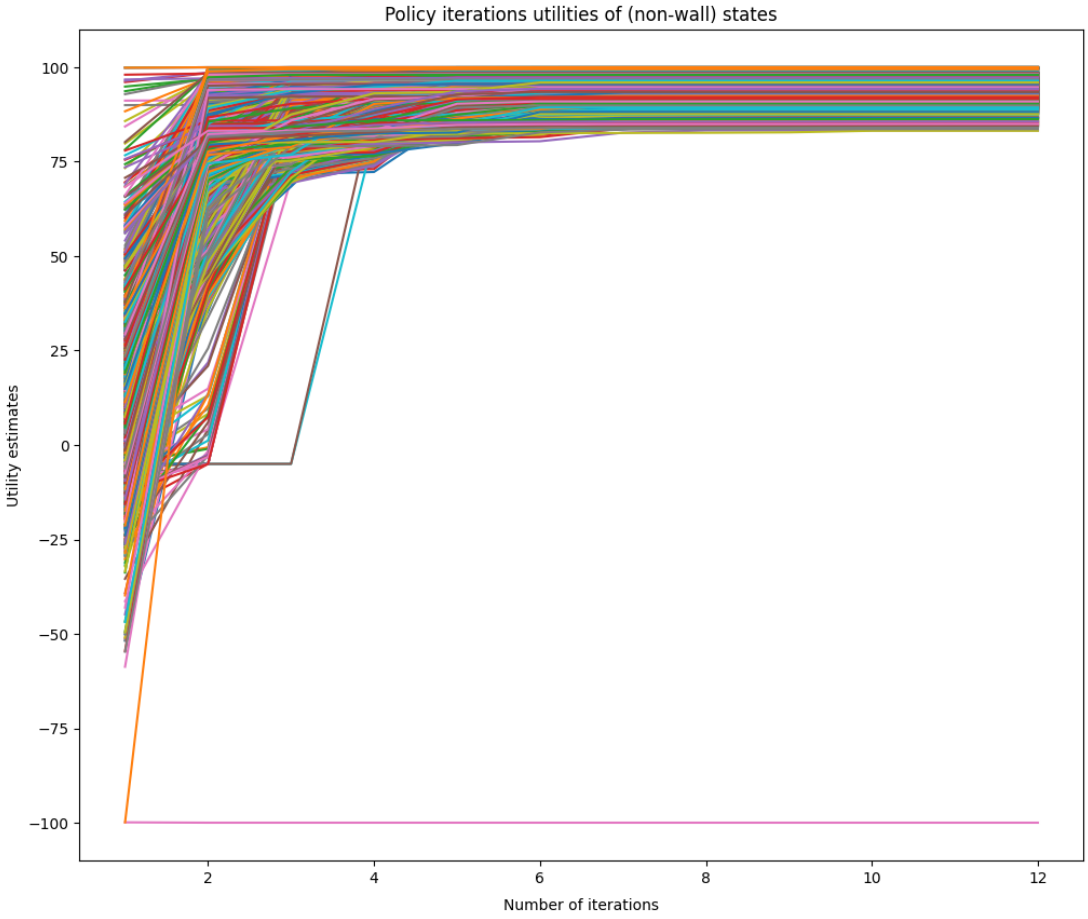


Figure 29: Policy iteration utilities of all states

9.4 Plot of Policy Iteration utility estimates as a function of the number of iterations (wall omitted)



Note: Wall states are omitted

Figure 30: Plot of Policy Iteration utility estimates as a function of the number of iterations

Policy Iteration Observations

- Comparing given maze, 10x10 maze, and 50x50 maze

```
Given maze algorithm running results:
- There are 25 states in the maze
- Policy iteration converged in 6 iterations
- Time taken for convergence: 0.1769 seconds

10x10 maze algorithm running results:
- There are 100 states in the maze
- Policy iteration converged in 9 iterations
- Time taken for convergence: 1.0036 seconds

50x50 maze algorithm running results:
- There are 2,500 states in the maze
- Policy iteration converged in 12 iterations
- Time taken for convergence: 31.4488 seconds

Amplitude of 10x10_time_2 over given_time_1 is 5.67
Amplitude of 50x50_time_3 over 10x10_time_2 is 31.34
Amplitude of 50x50_time_3 over given_time_1 is 177.83
```

- The number of iterations for convergence grows as the number of states increases.

- The convergence time for the 50x50 maze is over 20 times longer than that of the 10x10 maze.

- The convergence time for the 50x50 maze is over 100 times longer than that of the given maze.

10. Testing with more complex maze environment

Complex Maze Environment Characteristics:

- A larger number of states within the maze.
- An increased wall-to-state ratio, indicating obstacle density.

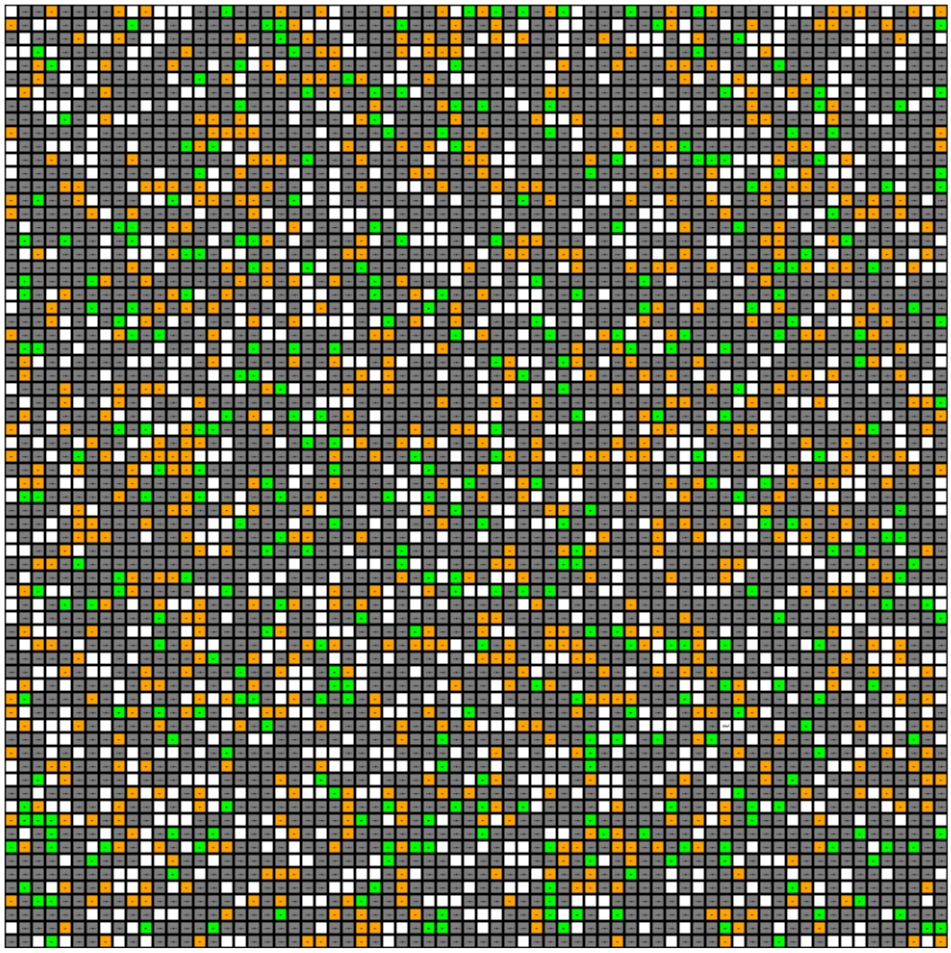
To identify a suitable complex maze for testing, I performed experiments with various maze sizes and obstacle densities using policy iteration. The time limit for each experiment was capped at **60 minutes** – any maze taking longer was stopped, and the time was recorded as **> 60 min**. The results are as follows:

Maze Size	Obstacle Density	Time Taken
50x50 maze	50% - 60%	29.1262 sec
50x50 maze	60% - 70%	21.6624 sec
60x60 maze	50% - 60%	3 min 24.32 sec
70x70 maze	50% - 60%	4 min 32.43 sec
80x80 maze	40% - 50%	> 60 min
80x80 maze	50% - 60%	> 60 min
90x90 maze	40% -560%	> 60 min
90x90 maze	50% - 60%	> 60 min
100x100 maze	50% - 60%	> 60 min

Table 4: Experiment results for maze sizes and obstacle densities with time taken for convergence

Based on these findings, I selected the 7**0x70 maze with a 50%-60% obstacle density** for further testing. It represented the largest maze with the highest density of obstacles that could be completed within the 60-minute benchmark.

A 80x80 maze with random configurations is generated (using a seed number for reproducible results):



```
# Maze size
MAZE_WIDTH  = 80
MAZE_HEIGHT = 80

# Starting position
START_POS = (78, 74)

# Green squares  – Count: 398
# Brown squares  – Count: 1041
# Wall locations – Count: 3665 (57.3)%
# White squares  – Count: 1296
```

10.1 Run the Value Iteration Algorithm

```
Value iteration converged in 1145 iterations
Time taken for convergence: 62.7827 seconds

Time taken for convergence: 1 min 2.78 sec
```

10.2 Plot of optimal Value Iteration policy

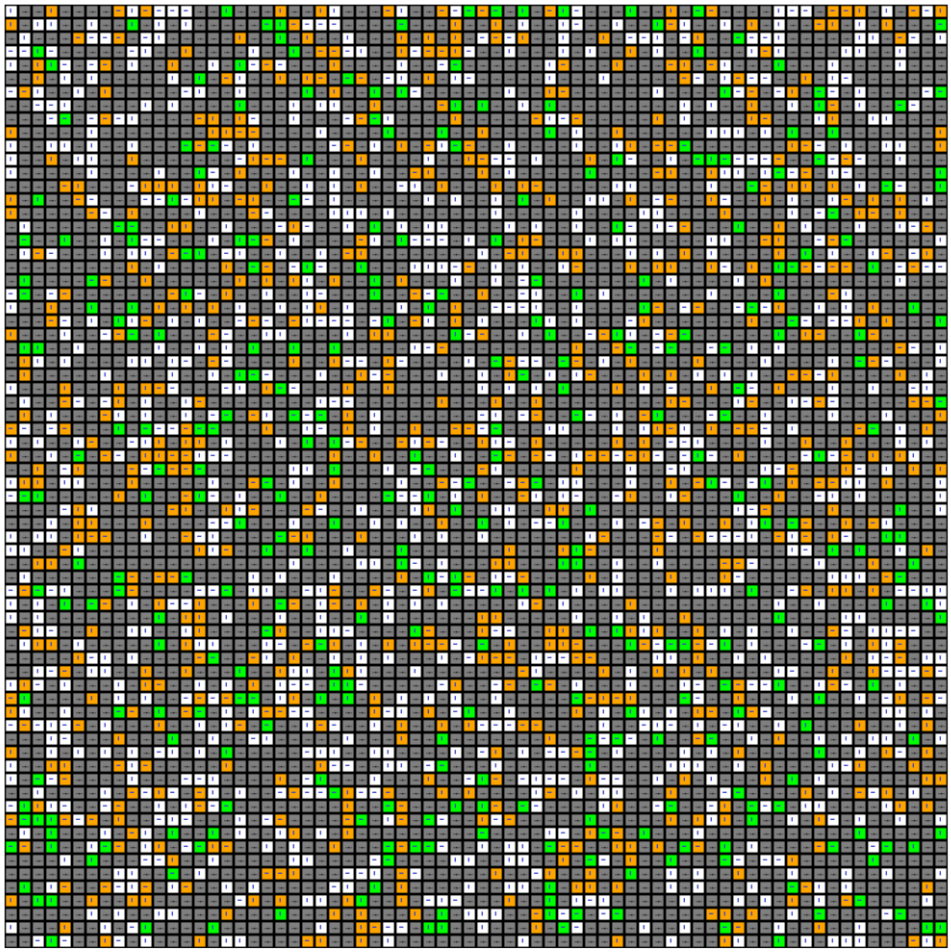


Figure 32: Plot of optimal Value Iteration policy

10.3 Value Iteration utilities of all states

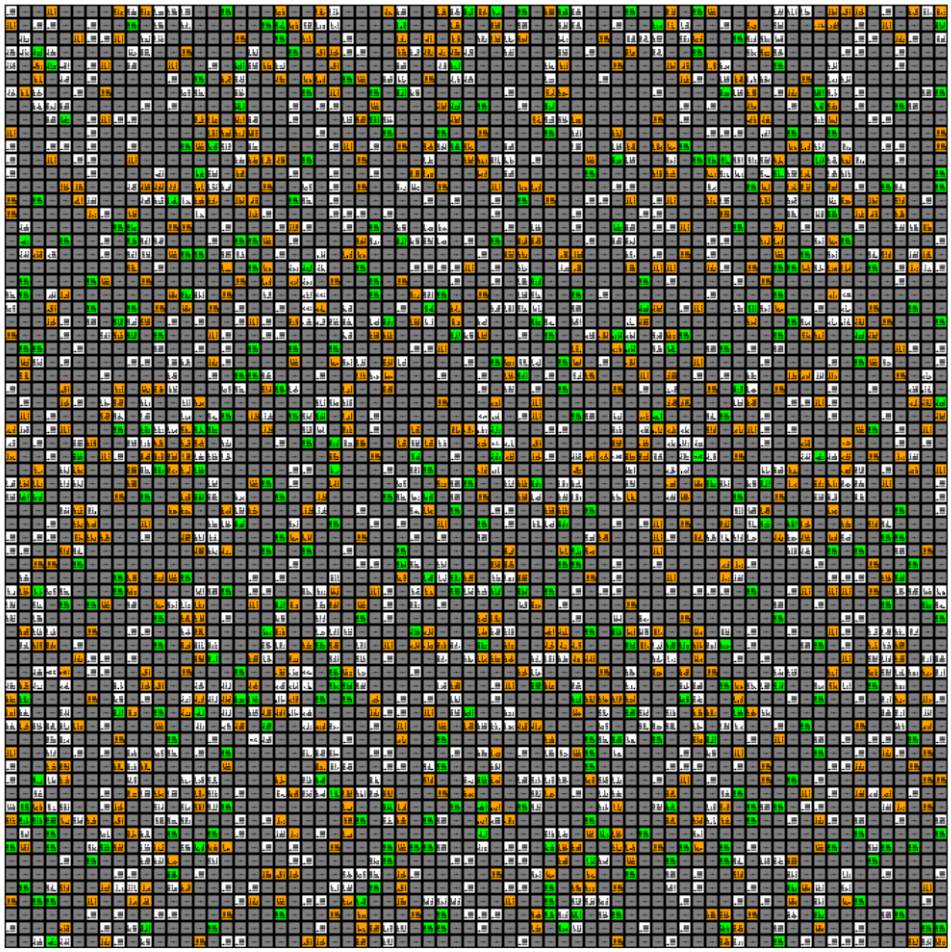


Figure 33: Value iteration utilities of all states

10.4 Plot of Value Iteration utility estimates as a function of the number of iterations (wall omitted)

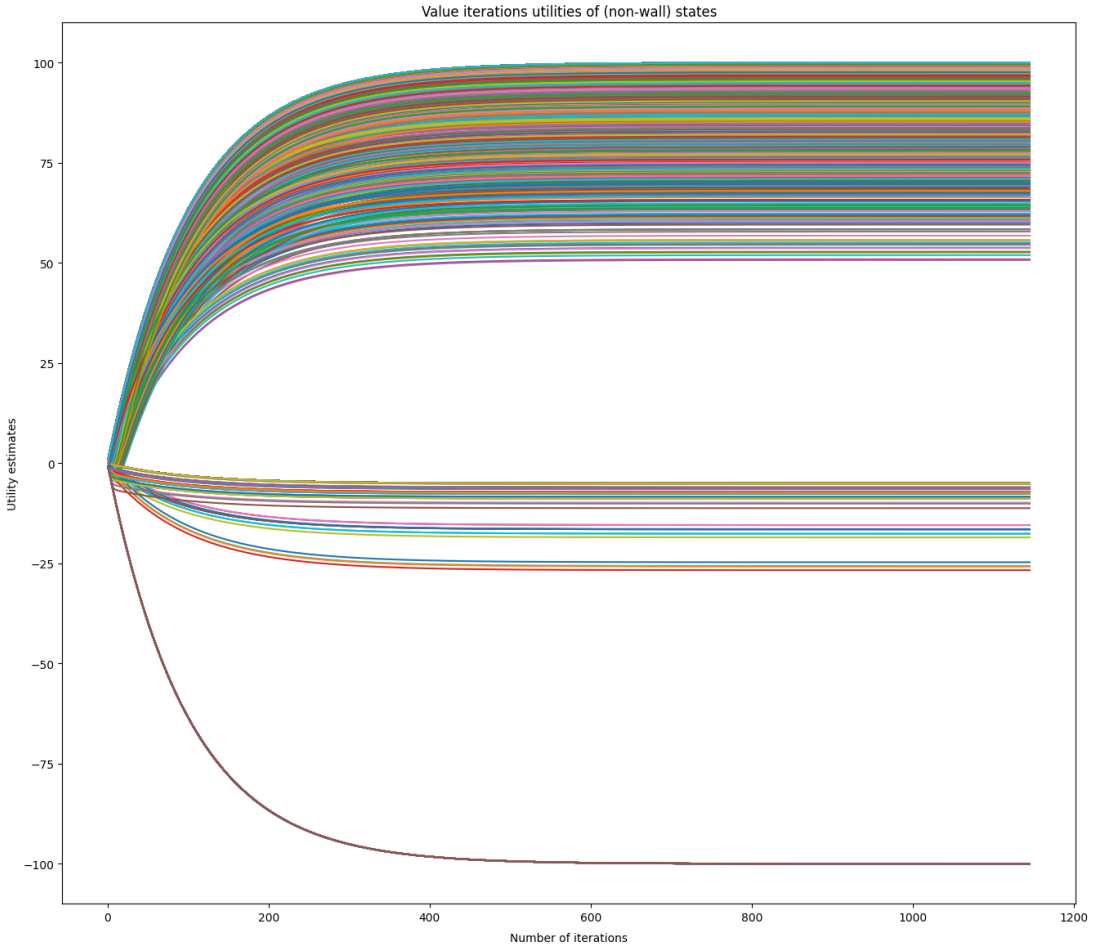


Figure 34: Plot of Value Iteration utility estimates as a function of the number of iterations

Value Iteration Observations

- Comparing given maze, 10x10 maze, 50x50, and 80x80 maze

```
Given maze algorithm running results:
- There are 25 states in the maze
- Value iteration converged in 1145 iterations
- Time taken for convergence: 0.7206 seconds

10x10 maze algorithm running results:
- There are 100 states in the maze
- Value iteration converged in 1145 iterations
- Time taken for convergence: 1.2521 seconds

50x50 maze algorithm running results:
- There are 2,500 states in the maze
- Value iteration converged in 1145 iterations
- Time taken for convergence: 32.6772 seconds

70x70 maze algorithm running results:
- There are 4,900 states in the maze
- Value iteration converged in 1145 iterations
- Time taken for convergence: 62.7827 seconds

Amplitude of 10x10_time_2 over given_time_1 is 1.74
Amplitude of 50x50_time_3 over 10x10_time_2 is 26.10
Amplitude of 50x50_time_3 over given_time_1 is 45.35
Amplitude of 70x70_time_4 over 50x50_time_3 is 1.92
Amplitude of 70x70_time_4 over 10x10_time_2 is 50.14
Amplitude of 70x70_time_4 over given_time_1 is 87.13
```

- The number of iterations for convergence is identical across all four maze sizes.

- The convergence time for the 70x80 maze is almost twice longer than that of the 50x50 maze.

- The convergence time for the 70x80 maze is over 40 times longer than that of the 10x10 maze.

- The convergence time for the 70x80 maze is over 80 times longer than that of the given maze.

11.1 Run the Policy Iteration Algorithm

(Using seed number for reproducible results)

```
Policy iteration converged in 42 iterations
Time taken for convergence: 307.5154 seconds

Time taken for convergence: 5 min 7.52 sec
```

11.2 Plot of optimal Policy Iteration policy

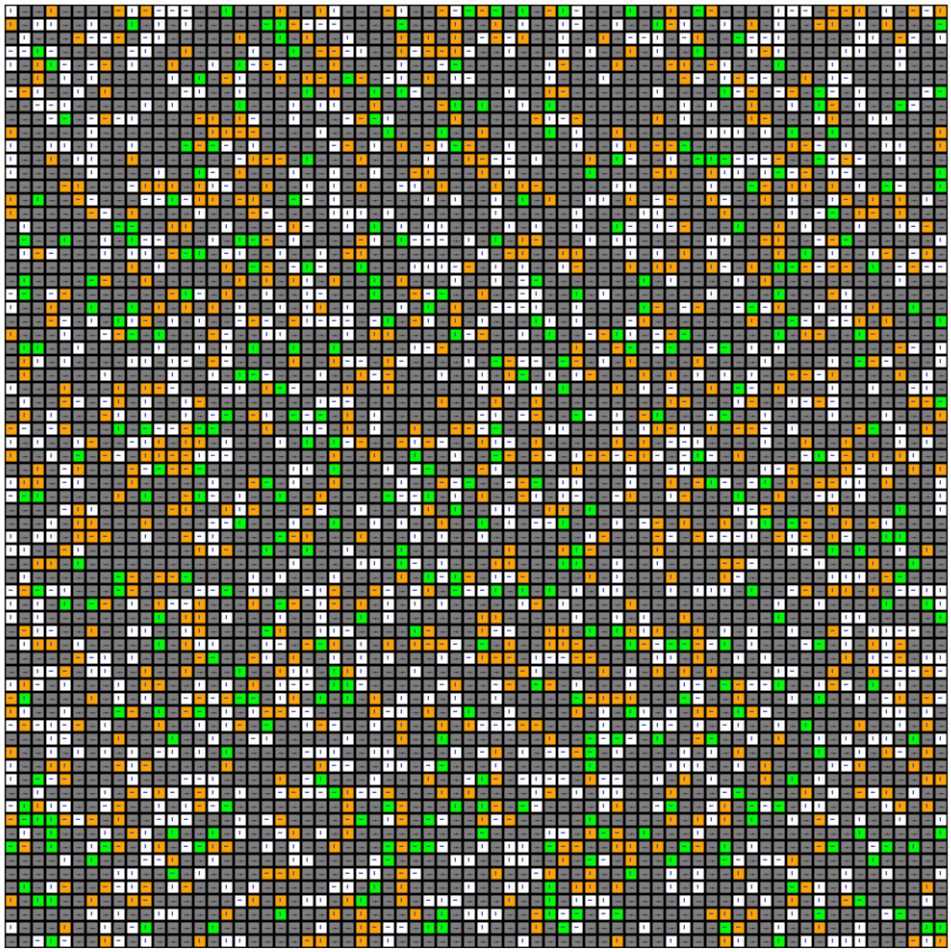


Figure 35: Plot of optimal Policy Iteration policy

11.3 Policy Iteration utilities of all states

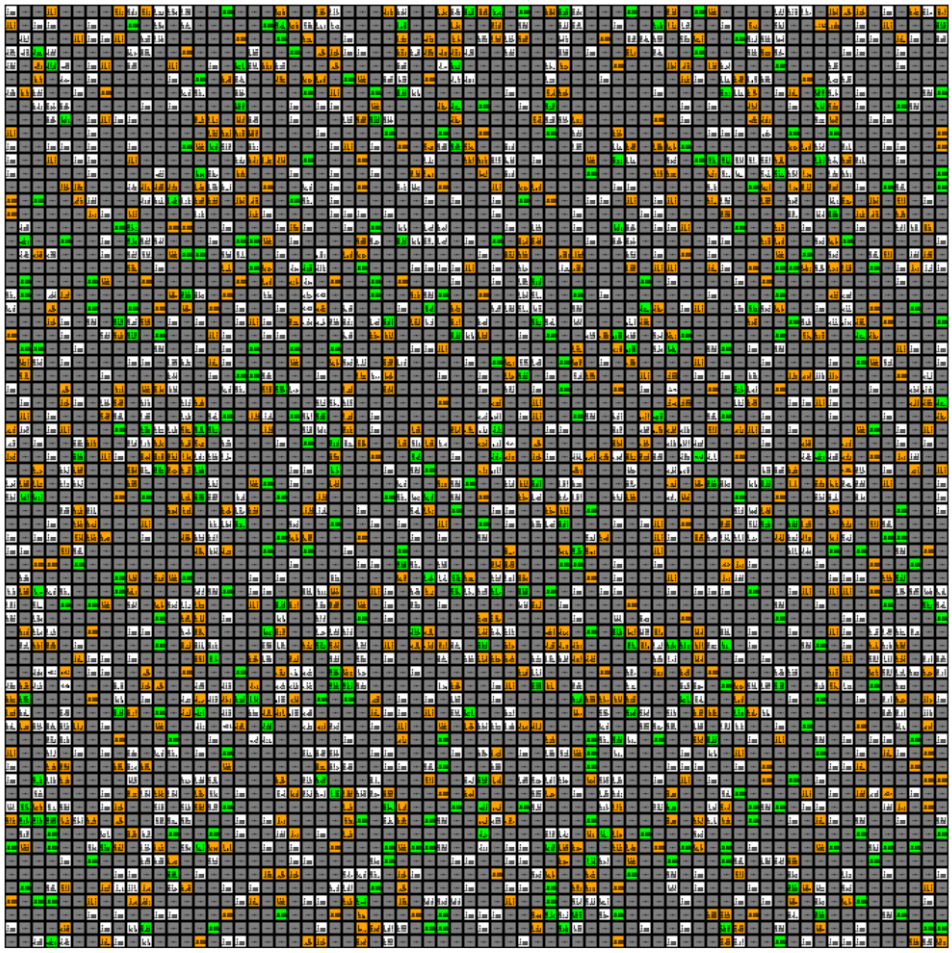


Figure 36: Policy iteration utilities of all states

11.4 Plot of Policy Iteration utility estimates as a function of the number of iterations (wall omitted)

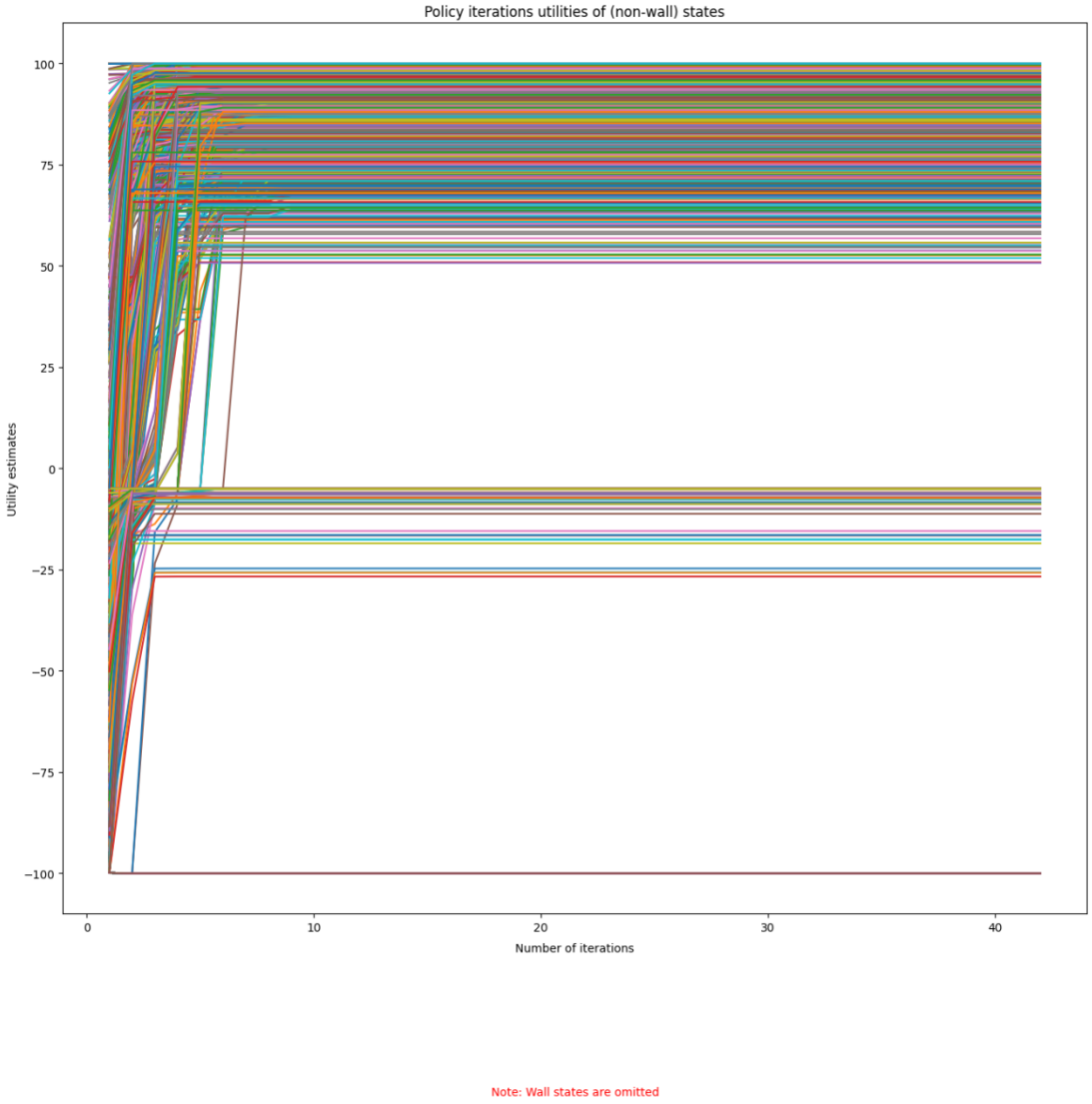


Figure 37: Plot of Policy Iteration utility estimates as a function of the number of iterations

Policy Iteration Observations

- Comparing given maze, 10x10 maze, 50x50 maze, and 70x70 maze

```
Given maze algorithm running results:
- There are 25 states in the maze
- Policy iteration converged in 6 iterations
- Time taken for convergence: 0.1769 seconds

10x10 maze algorithm running results:
- There are 100 states in the maze
- Policy iteration converged in 9 iterations
- Time taken for convergence: 1.0036 seconds

50x50 maze algorithm running results:
- There are 2,500 states in the maze
- Policy iteration converged in 12 iterations
- Time taken for convergence: 31.4488 seconds

70x70 maze algorithm running results:
- There are 4,900 states in the maze
- Policy iteration converged in 42 iterations
- Time taken for convergence: 307.5154 seconds

Amplitude of 10x10_time_2 over given_time_1 is 5.67
Amplitude of 50x50_time_3 over 10x10_time_2 is 31.34
Amplitude of 50x50_time_3 over given_time_1 is 177.83
Amplitude of 70x70_time_4 over 50x50_time_3 is 9.78
Amplitude of 70x70_time_4 over 10x10_time_2 is 306.41
Amplitude of 70x70_time_4 over given_time_3 is 1738.84
```

- The number of iterations for convergence grows as the number of states increases.

- The convergence time for the 70x70 maze is over 5 times longer than that of the 50x50 maze.

- The convergence time for the 70x70 maze is over 300 times longer than that of the 10x10 maze.

- The convergence time for the 70x70 maze is over 1,000 times longer than that of the given maze.

Plot number of iterations and convergence time vs number of states:

Value Iteration vs Policy Iteration

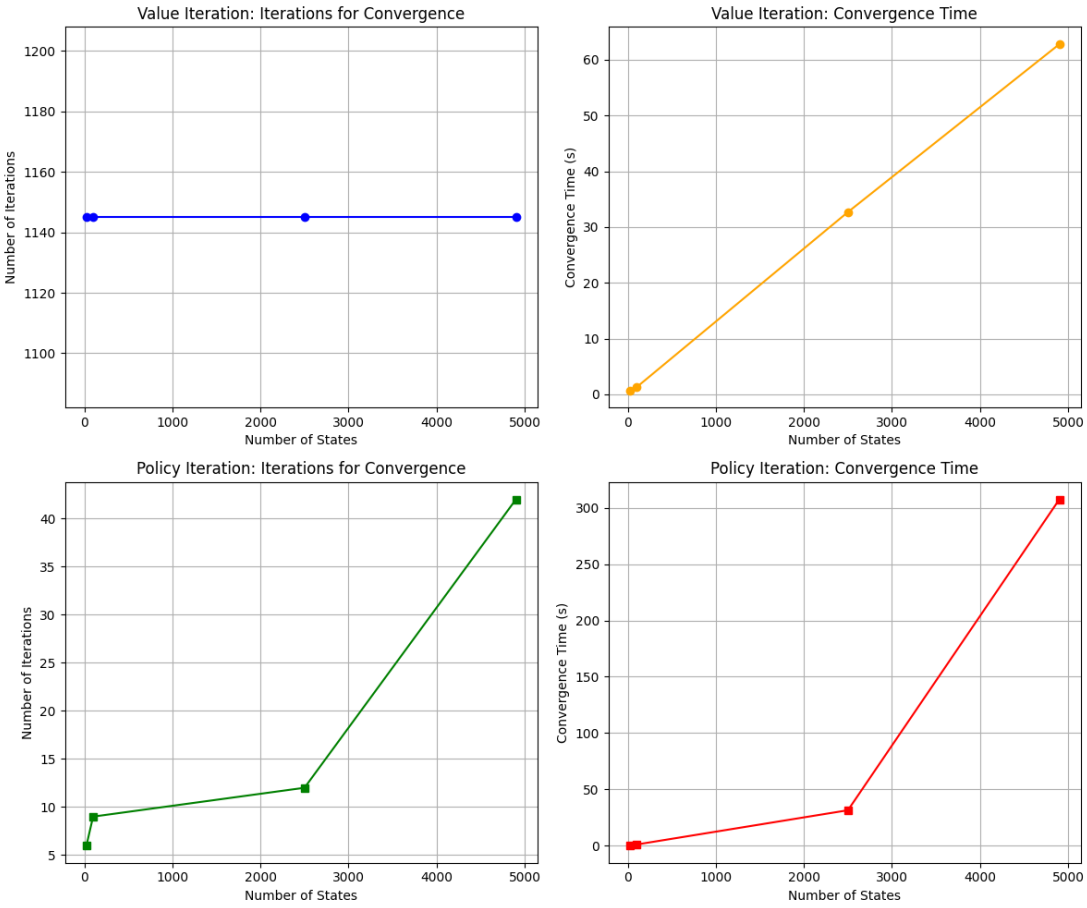


Figure 38: Plot number of iterations and convergence time vs number of states

12. Answer to Part 2 Questions

Question

Design a more complicated maze environment of your own and re-run the algorithms designed for Part 1 on it. How does the number of states and the complexity of the environment affect convergence?

Answer

Value Iteration

- As the number of states increases, the number of iterations required for value iteration to converge remains fixed. This indicates that value iteration has a stable convergence pattern, unaffected by the increase in the number of states.

- The convergence time for value iteration increases linearly as the number of states grows. This suggests that while the algorithm is efficient, larger state spaces will require proportionally more time.

Policy Iteration

- The number of iterations required for policy iteration to converge increases non-linearly with the number of states. This is because there are more state utility values to update and stabilize, leading to a more complex convergence process.

- With more states and a more complex environment, the computational complexity of each iteration in policy iteration increases exponentially. This can lead to significantly longer computation times for each iteration of policy evaluation and policy iteration.

Question

How complex can you make the environment and still be able to learn the right policy?

Answer

The limits of policy learning were evident in our experiments. As shown in Table 4 (page 35), the algorithms could only achieve optimal policies within a 70x70 maze with obstacle density between 50-60%. Increased maze size or obstacle complexity resulted in learning failures (within 60 min).

13. Conclusion

Both Value and Policy Iteration can still learn the optimal policy given enough time to converge, regardless of the number of states. However, as the environment becomes more complex, Policy Iteration may struggle with efficiency due to its exponential time growth, while Value Iteration remains more scalable for larger mazes.

For environments with a vast number of states, value iteration is likely to converge faster and more reliably than policy iteration. However, for smaller state spaces, policy iteration can sometimes converge in fewer iterations.