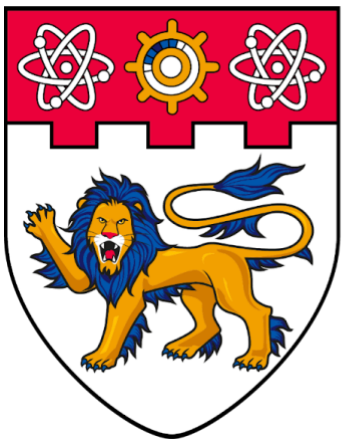

NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

CZ4046 INTELLIGENT AGENTS
Assignment 1

Submitted By: Soh Qian Yi
Matriculation Number: U1922306C

Table of Contents

1. The Bellman Equation

The Bellman equation is a fundamental concept in reinforcement learning that relates the utility *U* of a state *S* to the immediate reward obtained and the expected discounted utility of the next state, assuming the agent chooses the optimal action.

The Bellman equation is defined as:

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

Figure 1: Definition of the Bellman Equation

2. Part 1: Value Iteration

The Bellman equation is used in the value iteration algorithm to solve Markov Decision Processes (MDP). There are *n* Bellman equations, one for each state, which needs to be solved simultaneously to find the utilities of each state. An iterative approach is used, starting with initial values for the utilities and updating them until an equilibrium is reached. The utility value for state *s* at the i^{th} iteration is denoted as $U_i(s)$.

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

Figure 2: Definition of Bellman Update (Iteration Step)

function VALUE-ITERATION(mdp, ϵ) **returns** a utility function
  inputs: mdp, an MDP with states S, actions $A(s)$, transition model $P(s' \mid s, a)$,
    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**
      $U'[s] \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) \ U[s']$
      if $|U'[s] - U[s]| > \delta$ **then** $\delta \leftarrow |U'[s] - U[s]|$
  until $\delta < \epsilon(1-\gamma)/\gamma$
  return U

Figure 3: The Value Iteration Algorithm

2.1. Source Code Snippets

```python
def value_iteration(maze: Maze):
    """Value Iteration Algorithm Implementation"""

    termination_condition = max_error * \
        ((1 - DISCOUNT_FACTOR) / DISCOUNT_FACTOR)

    print("No. of Iterations: 0")      You, 50 seconds ago • Uncommitted changes
    maze.print_policy()
    maze.print_utility()
    csv_file = CSV("value_iteration", maze)
    csv_file.add_utilities(maze)

    # iterative approach with initial values for the utilities and updating them until an equilibrium is reached.
    i = 1
    while True:
        print(f"No. of Iterations: {i}")
        max_utility_change = 0

        # for each state
        for c in range(COLS):
            for r in range(ROWS):
                curr_cell = maze.get_cell(Coordinates(c, r))

                if curr_cell.get_cell_type() == Cells.Type.WALL:
                    continue

                # find maximum change in utility
                utility_change = calculate_utility(curr_cell, maze)
                if utility_change > max_utility_change:
                    max_utility_change = utility_change

        i += 1
        print(f"Max Utility Change: {max_utility_change:.3f}")
        maze.print_policy()
        maze.print_utility()

        csv_file.add_utilities(maze)

        # break out of loop if equilibrium is reached
        if max_utility_change < termination_condition:
            csv_file.write_csv()
            break
```

Figure 4: Code for value_iteration()

```python
def calculate_utility(curr_cell: Cells, maze: Maze):
    '''
    This function takes in two arguments:
    - curr_cell: an instance of the Cells class, representing the current cell
    - maze: an instance of the Maze class, representing the maze in which the current cell resides
    '''

    # Initialize a list of utilities for each direction
    utilities = [0.0] * 4

    # Iterate over all possible directions
    for direction in range(Coordinates.TOTAL_DIRECTIONS):
        neighbours = maze.get_neighbours_of_cell_direction(
            curr_cell, direction)

        # Calculate the weighted sum of the utilities of the neighboring cells in each direction
        up = P_UP * neighbours[0].get_utility()
        left = P_LEFT * neighbours[1].get_utility()
        right = P_RIGHT * neighbours[2].get_utility()

        utilities[direction] = up + left + right

    # Find the max utility
    max_utility = 0
    for i in range(1, len(utilities)):
        if utilities[i] > utilities[max_utility]:
            max_utility = i

    # Set new policy for state S
    curr_reward = curr_cell.get_reward(curr_cell.get_cell_type())
    curr_utility = curr_cell.get_utility()
    new_utility = curr_reward + DISCOUNT_FACTOR * utilities[max_utility]
    curr_cell.set_utility(new_utility)
    curr_cell.set_policy(max_utility)

    return abs(curr_utility - new_utility)
```

Figure 5: Code for calculate_utility()

In the above code snippets, the initial utility values of all states were initialized to 0, and their actions to *'Up'*.

During each iteration, for each state, the algorithm calculates the maximum expected utility for its next state using the Bellman Equation. It then determines the maximum change in utility for any state during this iteration. The process is repeated until the maximum change in utility across all states is below the calculated value for termination condition. The algorithm stops once this condition is met and the optimal policy can then be determined based on the updated utility values.

With reference to Figure 3, the termination condition is denoted as $\delta < \epsilon(1-\gamma)/\gamma$, where ϵ is the maximum error allowed in the utility of any state, and γ is the discount factor. In Figure 4, the termination condition is defined as $\epsilon(1-\gamma)/\gamma$.

For this assignment, ϵ was defined to be a value of 68, and γ was defined as the value 0.99.

2.2. Results

In this report, a maximum error value ϵ was defined as 68 for the following results.

2.2.1. Plot of Optimal Policy

The following Figure 6 shows the initial plot before the first iteration. Policies were initialized to *'Up'*.

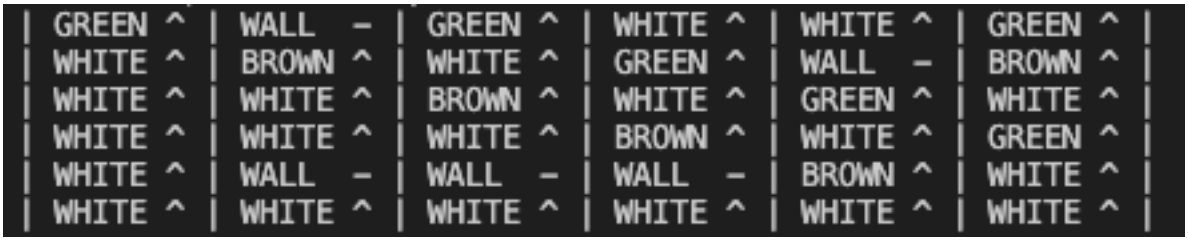

Figure 6: Plot of Optimal Policy at Iteration 0 (Value Iteration)

After convergence on the 38th iteration, Figure 7 shows the final plot of the optimal policies of all states.

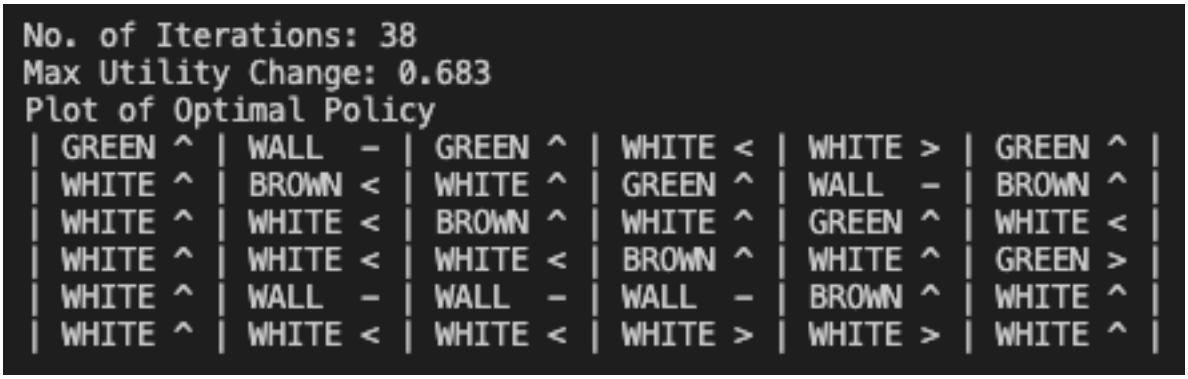

Figure 7: Plot of Optimal Policy at Iteration 38 (Value Iteration)

2.2.2. Utilities of all States

The following Figure 8 shows the initial utilities value before the first iteration.

```
Utilities of all States
| GREEN   1.000 | WALL    0.000 | GREEN   1.000 | WHITE  -0.040 | WHITE  -0.040 | GREEN   1.000 |
| WHITE  -0.040 | BROWN  -1.000 | WHITE  -0.040 | GREEN   1.000 | WALL    0.000 | BROWN  -1.000 |
| WHITE  -0.040 | WHITE  -0.040 | BROWN  -1.000 | WHITE  -0.040 | GREEN   1.000 | WHITE  -0.040 |
| WHITE  -0.040 | WHITE  -0.040 | WHITE  -0.040 | BROWN  -1.000 | WHITE  -0.040 | GREEN   1.000 |
| WHITE  -0.040 | WALL    0.000 | WALL    0.000 | WALL    0.000 | BROWN  -1.000 | WHITE  -0.040 |
| WHITE  -0.040 | WHITE  -0.040 | WHITE  -0.040 | WHITE  -0.040 | WHITE  -0.040 | WHITE  -0.040 |
```

Figure 8: Utilities at all States at Iteration 0

After convergence on the 38th iteration, Figure 9 shows the final utilities at all states.

```
Utilities of all States
| GREEN  32.427 | WALL    0.000 | GREEN  31.218 | WHITE  30.709 | WHITE  30.149 | GREEN  31.374 |
| WHITE  31.591 | BROWN  29.841 | WHITE  30.686 | GREEN  31.406 | WALL    0.000 | BROWN  29.580 |
| WHITE  30.906 | WHITE  30.278 | BROWN  29.288 | WHITE  30.795 | GREEN  31.545 | WHITE  30.947 |
| WHITE  30.253 | WHITE  29.804 | WHITE  29.301 | BROWN  29.290 | WHITE  30.918 | GREEN  31.668 |
| WHITE  29.665 | WALL    0.000 | WALL    0.000 | WALL    0.000 | BROWN  29.336 | WHITE  30.949 |
| WHITE  29.016 | WHITE  28.447 | WHITE  27.886 | WHITE  28.081 | WHITE  29.231 | WHITE  30.307 |
```

Figure 9: Utilities at all States at Iteration 38

2.3. Plot of Utility Estimates as a function of No. of Iterations

For the remainder of the report, the following code snippet will be used to generate the plot of Utility Estimates as a function of No. of Iterations. The *MinMaxScaler* function from the *sklearn* library is utilized to obtain normalized values for the Utility Estimates. This normalization enables a clearer representation of the changes in utility values across different iterations.

```python
import os
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler

path = '../results'
csv_files = [os.path.join(path, file) for file in os.listdir(path) if file.endswith('.csv')]

# Loop through each CSV file and generate the plot
for file in csv_files:
    # Load the data into a Pandas DataFrame
    data = pd.read_csv(file)

    # Normalize each column in the DataFrame
    scaler = MinMaxScaler()
    data_normalized = pd.DataFrame(scaler.fit_transform(data), columns=data.columns)

    # Plot each column as a separate line plot using the normalized values
    data_normalized.plot(kind='line', figsize=(12,8))

    # Set plot title and axis labels
    plt.title(f'Utility Estimates vs No. of Iterations ({os.path.splitext(os.path.basename(file))[0]})')
    plt.xlabel('No. of Iterations')
    plt.ylabel('Normalized Utility Estimates')

    plt.legend(loc='upper center', bbox_to_anchor=(0.5, -0.2), ncol=6, fontsize='medium', title='States', title_fontsiz

    # Save the plot as a PNG image
    plot_name = os.path.splitext(os.path.basename(file))[0] + '.png'
    plt.savefig(plot_name, bbox_inches='tight')
    plt.show()


    # Clear the current plot and move on to the next file
    plt.clf()
```

Figure 10: Code for Plots (plots.ipynb)

The following Figure 11 shows the plot of Utility Estimates as a function of No. of Iterations. From this figure, it is observed that value iteration converges at the 38th iteration.

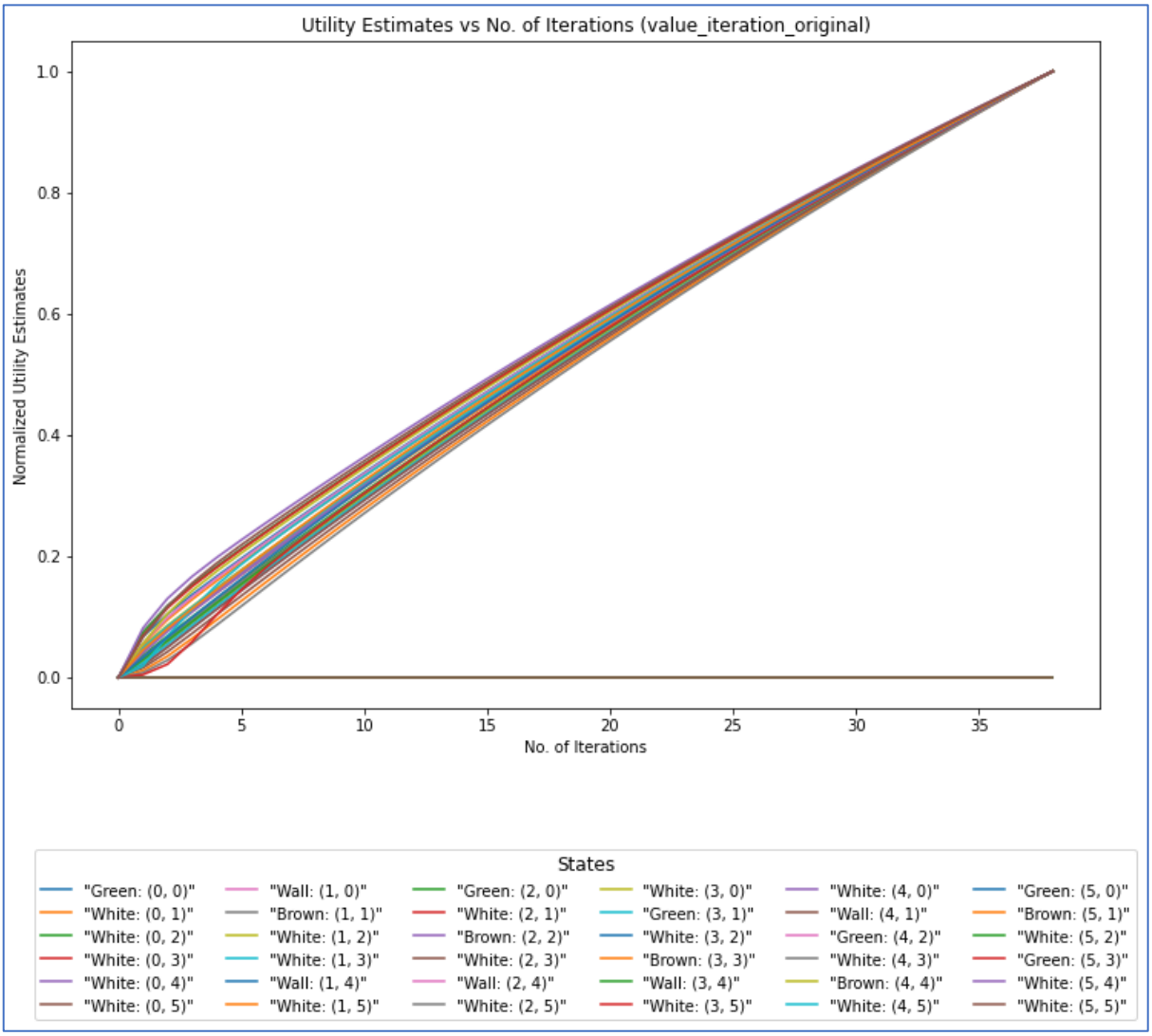

Figure 11: Utility Estimates vs No. of Iterations (value_iteration_original)

3. Part 1: Policy Iteration

Policy iteration is another algorithm used to solve Markov Decision Processes (MDP) that involves iteratively improving the policy and estimating its value. The algorithm consists of two main steps: policy evaluation and policy improvement.

In policy evaluation, the algorithm estimates the value of the current policy by solving the Bellman equation for the state-value function and once the value of the current policy has been estimated, policy improvement is performed by selecting the action that maximizes the expected value of the next state. These two steps are repeated iteratively until the policy converges to an optimal policy and yields no change in the utilities.

For MDPs with a large number of states, a simplified Bellman update (Figure 12) can be utilized in the policy evaluation phase. To obtain the next estimate of the utility, the simplified update is applied iteratively for a fixed number of times, eg. when k = 4.

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

Figure 12: Simplified Bellman Update Equation

function POLICY-ITERATION(mdp) **returns** a policy
  inputs: mdp, an MDP with states S, actions $A(s)$, transition model $P(s' \mid s, a)$
  local variables: U, a vector of utilities for states in S, initially zero
    π, a policy vector indexed by state, initially random

  repeat
    $U \leftarrow$ POLICY-EVALUATION(π, U, mdp)
    $unchanged? \leftarrow$ true
    for each state s **in** S **do**
      if $\max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\ U[s'] > \sum_{s'} P(s' \mid s, \pi[s])\ U[s']$ **then do**
        $\pi[s] \leftarrow \operatorname{argmax}_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\ U[s']$
        $unchanged? \leftarrow$ false
  until $unchanged?$
  return π

Figure 13: The Policy Iteration Algorithm

3.1. Source Code Snippets

```python
def policy_iteration(maze: Maze):
    """Policy Iteration Algorithm Implementation"""

    print("No. of Iterations: 0")
    maze.print_policy()
    maze.print_utility()
    csv_file = CSV("policy_iteration", maze)
    csv_file.add_utilities(maze)
    i = 1

    # repeat until the policy converges
    while True:
        print(f"No. of Iterations: {i}")

        # set a flag to check if the policy has changed during the iteration
        unchanged = True
        # call the policy_evaluation function to estimate the utility values
        policy_evaluation(maze, k)

        # iterate over each cell in the maze to improve the policy
        for c in range(COLS):
            for r in range(ROWS):
                curr_cell = maze.get_cell(Coordinates(c, r))

                if curr_cell.get_cell_type() == Cells.Type.WALL:
                    continue
                # call the policy_improvement function to improve the policy for the current cell
                # if the policy has changed, set the flag to False
                if policy_improvement(curr_cell, maze):
                    unchanged = False

        maze.print_policy()
        maze.print_utility()
        csv_file.add_utilities(maze)

        # if the policy has not changed during this iteration, break the loop
        if unchanged:
            break

        i += 1

    csv_file.write_csv()
```

Figure 14: Code for policy_iteration()

```python
def policy_evaluation(maze: Maze, k: int):
    '''
    Args:
    - maze (Maze): the maze environment
    - k (int): number of iterations for policy evaluation

    Returns:
    - None
    '''
    for _ in range(k):
        for c in range(COLS):
            for r in range(ROWS):
                curr_cell = maze.get_cell(Coordinates(c, r))

                if curr_cell.get_cell_type() == Cells.Type.WALL:
                    continue

                # Calculate the utility of the current state using Bellman update and current policy
                neighbours = maze.get_neighbours_of_cell_current_policy(
                    curr_cell)
                up = P_UP * neighbours[0].get_utility()
                left = P_LEFT * neighbours[1].get_utility()
                right = P_RIGHT * neighbours[2].get_utility()

                reward = curr_cell.get_reward(curr_cell.cell_type)

                utility = reward + DISCOUNT_FACTOR * (up + left + right)
                curr_cell.set_utility(utility)
```

Figure 15: Code for policy_evaluation()

```python
def policy_improvement(curr_cell: Cells, maze: Maze):
    """
    Given the current state (curr_cell) and the current policy, updates the policy
    if a better policy is found.

    Args:
    - curr_cell: the current cell whose policy is being improved
    - maze: the maze object

    Returns:
    - True if the policy is updated, False otherwise
    """

    max_utility = [0.0] * 4

    # determine the maximum expected utilities from the neighboring cells
    for direction in range(Coordinates.TOTAL_DIRECTIONS):
        neighbours = maze.get_neighbours_of_cell_direction(
            curr_cell, direction)
        up = P_UP * neighbours[0].get_utility()
        left = P_LEFT * neighbours[1].get_utility()
        right = P_RIGHT * neighbours[2].get_utility()

        max_utility[direction] = up + left + right

    # get max utility
    max_direction = max_utility.index(max(max_utility))

    # calculate the utility of a state using current policy
    neighbours = maze.get_neighbours_of_cell_current_policy(curr_cell)
    up = P_UP * neighbours[0].get_utility()
    left = P_LEFT * neighbours[1].get_utility()
    right = P_RIGHT * neighbours[2].get_utility()

    curr_utility = up + left + right

    if max_utility[max_direction] > curr_utility:
        curr_cell.set_policy(max_direction)
        return True
    else:
        return False
```

Figure 16: Code for policy_improvement()

In the above code snippets, the initial utility values of all states were initialized to 0, and their actions to *'Up'*, and initialize a *unchanged* Boolean to check if policy has changed during the iteration. For each state, policy evaluation is performed for *k* times (in this case, *k* was defined as 4) and policy improvement is performed. Policy improvement returns true if any policy is updated. This iterative process is continued until the policy reaches an optimal state and there are no further changes in the utilities.

3.2. Results

In this report, *k* value was defined as 4 for the following results.

3.2.1. Plot of Optimal Policy

The following Figure 17 shows the initial plot before the first iteration. Policies were initialized to *'Up'*.

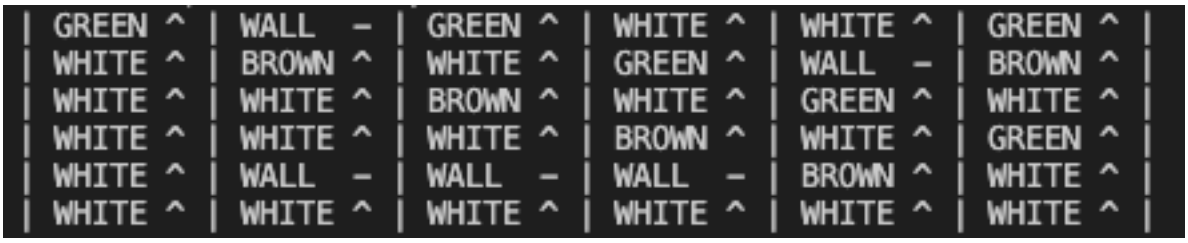

Figure 17: Plot of Optimal Policy at Iteration 0 (Policy Iteration)

After convergence on the 10th iteration, Figure 18 shows the final plot of the optimal policies of all states. It is observed that the policy iteration algorithm requires less iterations to obtain the optimal policy as compared to the valuation iteration algorithm.

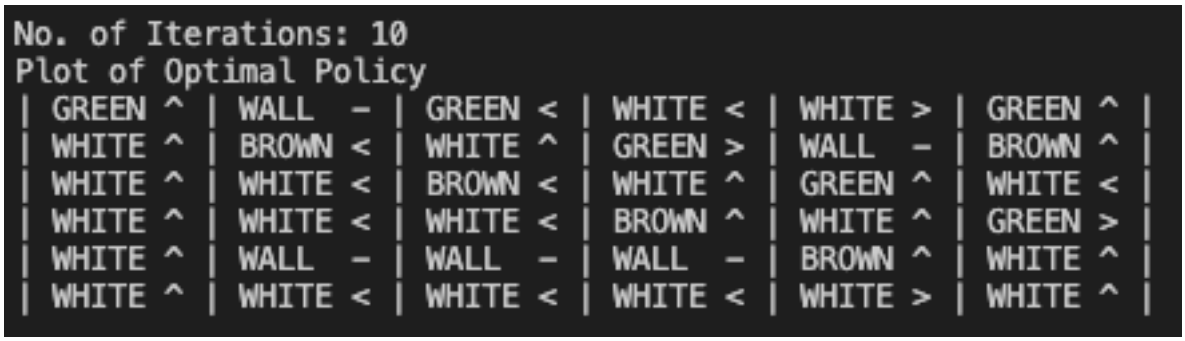

Figure 18: Plot of Optimal Policy at Iteration 10 (Policy Iteration)

3.2.2. Utilities of all States

The following Figure 19 shows the initial utilities value before the first iteration.

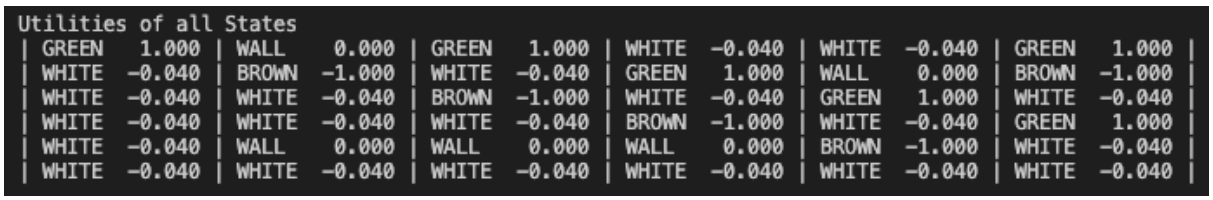

Figure 19: Utilities at all States at Iteration 0 (Policy Iteration)

After convergence on the 10th iteration, the following Figure 20 shows the final utilities at all states.

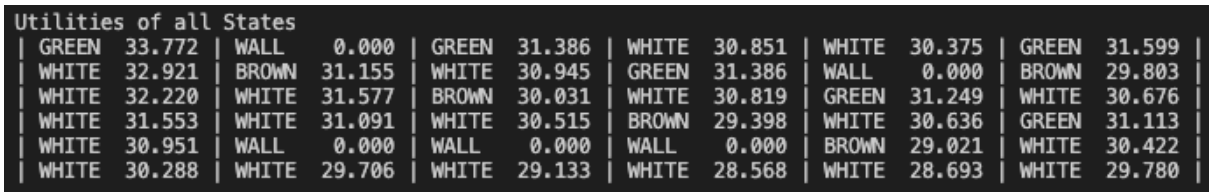

Figure 20: Utilities at all States at Iteration 10 (Policy Iteration)

3.3. Plot of Utility Estimates as a function of No. of Iterations

The following Figure 21 shows the plot of Utility Estimates as a function of No. of Iterations. From this figure, it is observed that policy iteration converges at the 10th iteration.

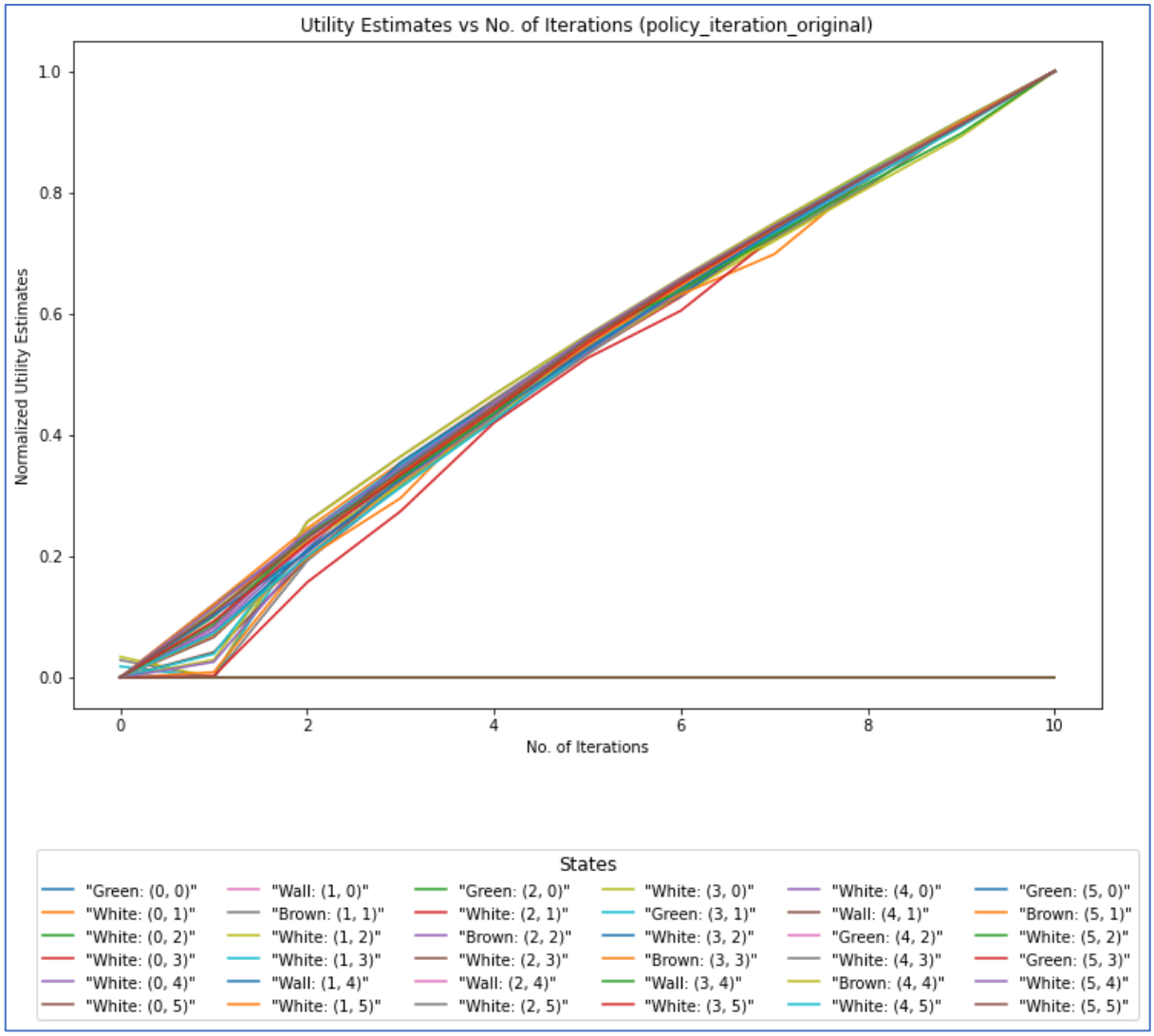

Figure 21: Utility Estimates vs No. of Iterations (policy_iteration_original)

4. Part 2: Bonus Questions

This section delves into how the complexity of the environment and the number of states impact the number of convergence iterations for both value iteration and policy iteration methods compared to the original maze (Figure 22) provided for this assignment.

Figure 22: Original Maze

Legend:
G – Green Squares
W – White Squares
B – Brown Squares
- Walls

4.1. Increasing the Walls of Maze

In the following maze (Figure 23), there are more walls *(#)* than in the original maze (Figure 22) from Part 1, while the size of the maze (number of rows and columns) were kept constant.

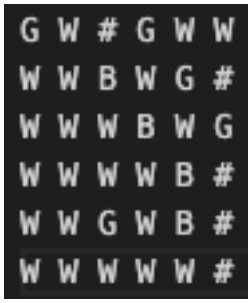

Figure 23: Maze with Increased Wall States

4.1.1. Results

4.1.1.1. Value Iteration

Figure 24 shows the final plot of the optimal policies and utilities of all states, where it is observed that the number of iterations before convergence for value iteration decreased by one.

```
No. of Iterations: 37
Max Utility Change: 0.681
Plot of Optimal Policy
| GREEN ^ | WHITE < | WALL  - | GREEN ^ | WHITE < | WHITE < |
| WHITE ^ | WHITE < | BROWN > | WHITE ^ | GREEN > | WALL  - |
| WHITE ^ | WHITE ^ | WHITE < | BROWN > | WHITE > | GREEN > |
| WHITE ^ | WHITE ^ | WHITE < | WHITE ^ | BROWN ^ | WALL  - |
| WHITE ^ | WHITE ^ | GREEN ^ | WHITE ^ | BROWN ^ | WALL  - |
| WHITE ^ | WHITE ^ | WHITE ^ | WHITE ^ | WHITE ^ | WALL  - |

Utilities of all States
| GREEN  30.156 | WHITE  29.513 | WALL    0.000 | GREEN  30.598 | WHITE  30.129 | WHITE  29.544 |
| WHITE  29.513 | WHITE  29.016 | BROWN  27.570 | WHITE  29.940 | GREEN  31.132 | WALL    0.000 |
| WHITE  28.893 | WHITE  28.507 | WHITE  27.945 | BROWN  28.837 | WHITE  31.143 | GREEN  32.582 |
| WHITE  28.296 | WHITE  28.017 | WHITE  27.663 | WHITE  28.373 | BROWN  29.310 | WALL    0.000 |
| WHITE  27.731 | WHITE  27.642 | GREEN  28.345 | WHITE  27.910 | BROWN  27.648 | WALL    0.000 |
| WHITE  27.192 | WHITE  27.233 | WHITE  27.760 | WHITE  27.442 | WHITE  27.203 | WALL    0.000 |
```

Figure 24: Results for Value Iteration at Iteration 37

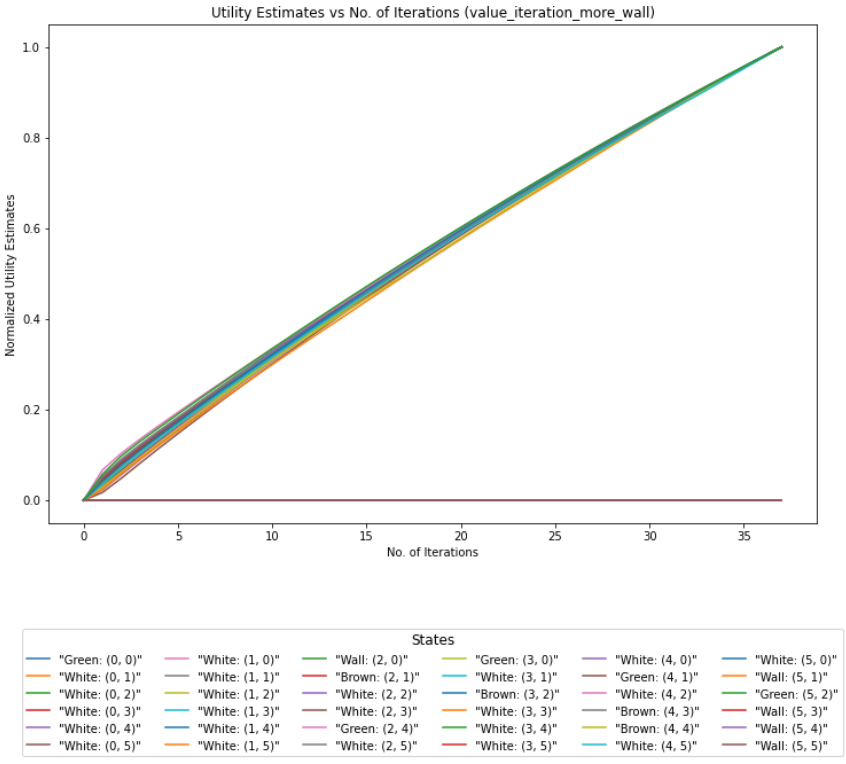

Figure 25: Utility Estimates vs No. of Iterations (value_iteration_more_wall)

4.1.1.2. Policy Iteration

Figure 26 shows the final plot of the optimal policies and utilities of all states, where it is observed that the number of iterations before convergence for policy iteration increased from 10 iterations to 16 iterations, indicating that the additional walls may have increased the complexity of the environment and the number of states that the agent must consider, and hence slowed down convergence.

```
No. of Iterations: 16
Plot of Optimal Policy
| GREEN ^ | WHITE < | WALL  - | GREEN ^ | WHITE v | WHITE < |
| WHITE ^ | WHITE < | BROWN > | WHITE > | GREEN v | WALL  - |
| WHITE ^ | WHITE ^ | WHITE > | BROWN > | WHITE > | GREEN > |
| WHITE ^ | WHITE ^ | WHITE > | WHITE > | BROWN ^ | WALL  - |
| WHITE ^ | WHITE > | GREEN > | WHITE ^ | BROWN ^ | WALL  - |
| WHITE ^ | WHITE > | WHITE ^ | WHITE ^ | WHITE ^ | WALL  - |

Utilities of all States
| GREEN  44.712 | WHITE  43.908 | WALL    0.000 | GREEN  45.268 | WHITE  44.966 | WHITE  44.228 |
| WHITE  43.908 | WHITE  43.259 | BROWN  42.500 | WHITE  44.935 | GREEN  46.229 | WALL    0.000 |
| WHITE  43.136 | WHITE  42.662 | WHITE  42.651 | BROWN  43.921 | WHITE  46.303 | GREEN  47.794 |
| WHITE  42.392 | WHITE  42.059 | WHITE  42.069 | WHITE  43.112 | BROWN  44.271 | WALL    0.000 |
| WHITE  41.653 | WHITE  41.323 | GREEN  42.462 | WHITE  42.457 | BROWN  42.414 | WALL    0.000 |
| WHITE  40.931 | WHITE  40.654 | WHITE  41.703 | WHITE  41.800 | WHITE  41.776 | WALL    0.000 |
```

Figure 26: Results for Policy Iteration at Iteration 16

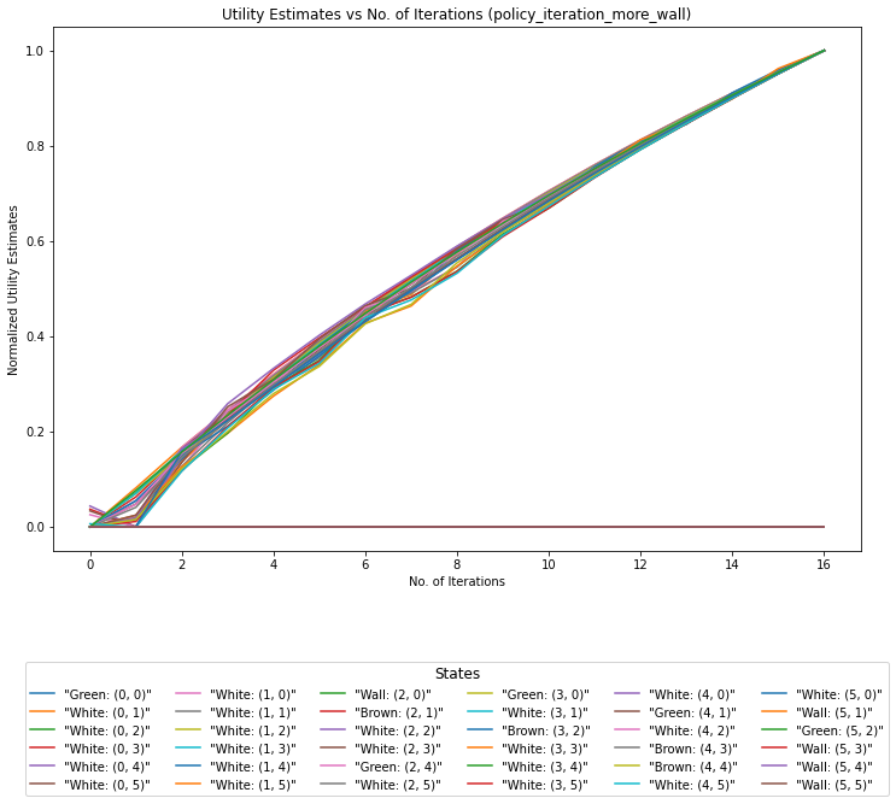

Figure 27: Utility Estimates vs No. of Iterations (policy_iteration_more_wall)

4.2. Increasing the Green States of Maze

In the following maze (Figure 28), there are more green states than in the original maze from Part 1 (Figure 22), while the size of the maze (number of rows and columns) were kept constant.

Figure 28: Maze with Increased Green States

4.2.1. Results

4.2.1.1. Value Iteration

Figure 29 shows the final plot of the optimal policies and utilities of all states, where it is observed that the number of iterations before convergence for value iteration decreased by one.

```
No. of Iterations: 37
Max Utility Change: 0.681
Plot of Optimal Policy
| GREEN ^ | WHITE < | WALL  - | GREEN ^ | WHITE < | WHITE < |
| WHITE ^ | WHITE < | BROWN > | WHITE ^ | GREEN > | WALL  - |
| WHITE ^ | WHITE < | WHITE < | BROWN > | WHITE > | GREEN > |
| GREEN < | WHITE < | WHITE < | WHITE ^ | BROWN ^ | WALL  - |
| WHITE ^ | WHITE ^ | GREEN < | WHITE < | BROWN v | WALL  - |
| WHITE ^ | WHITE ^ | WHITE ^ | WHITE > | GREEN v | WALL  - |

Utilities of all States
| GREEN  30.157 | WHITE  29.514 | WALL    0.000 | GREEN  30.599 | WHITE  30.130 | WHITE  29.545 |
| WHITE  29.514 | WHITE  29.022 | BROWN  27.587 | WHITE  29.942 | GREEN  31.134 | WALL    0.000 |
| WHITE  28.899 | WHITE  28.548 | WHITE  28.082 | BROWN  28.850 | WHITE  31.145 | GREEN  32.582 |
| GREEN  29.787 | WHITE  29.170 | WHITE  28.701 | WHITE  28.488 | BROWN  29.324 | WALL    0.000 |
| WHITE  29.171 | WHITE  28.808 | GREEN  29.471 | WHITE  29.025 | BROWN  28.891 | WALL    0.000 |
| WHITE  28.587 | WHITE  28.420 | WHITE  29.018 | WHITE  29.957 | GREEN  31.213 | WALL    0.000 |
```

Figure 29: Results for Value Iteration at Iteration 37

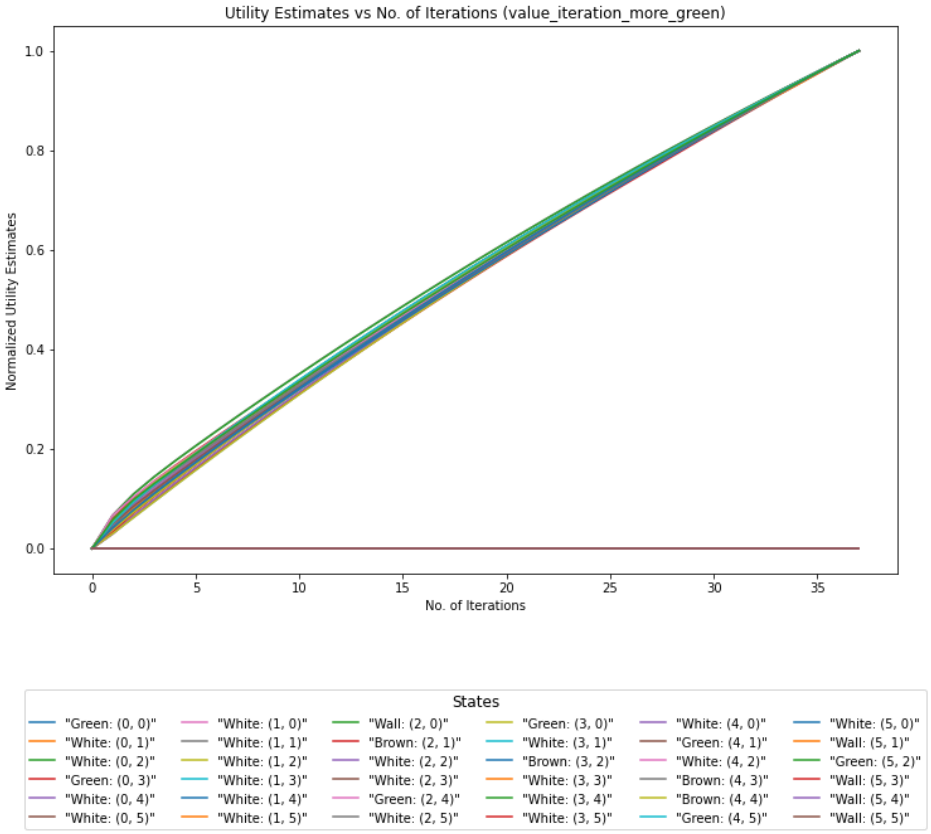

Figure 30: Utility Estimates vs No. of Iterations (value_iteration_more_green)

4.2.1.2. Policy Iteration

Figure 31 shows the final plot of the optimal policies and utilities of all states, where it is observed that the number of iterations before convergence for policy iteration increased from 10 iterations to 19 iterations, indicating that the additional green squares may have increased the number of possible actions the agent can take at each state, and hence slowed down convergence.

```
No. of Iterations: 19
Plot of Optimal Policy
| GREEN ^ | WHITE < | WALL  - | GREEN ^ | WHITE v | WHITE < |
| WHITE ^ | WHITE ^ | BROWN > | WHITE > | GREEN v | WALL  - |
| WHITE ^ | WHITE ^ | WHITE > | BROWN > | WHITE > | GREEN > |
| GREEN < | WHITE < | WHITE > | WHITE > | BROWN ^ | WALL  - |
| WHITE ^ | WHITE ^ | GREEN > | WHITE ^ | BROWN ^ | WALL  - |
| WHITE ^ | WHITE ^ | WHITE > | WHITE > | GREEN v | WALL  - |

Utilities of all States
| GREEN  50.053 | WHITE  49.195 | WALL    0.000 | GREEN  51.152 | WHITE  50.857 | WHITE  50.059 |
| WHITE  49.195 | WHITE  48.538 | BROWN  48.396 | WHITE  50.832 | GREEN  52.135 | WALL    0.000 |
| WHITE  48.377 | WHITE  47.950 | WHITE  48.537 | BROWN  49.820 | WHITE  52.218 | GREEN  53.725 |
| GREEN  49.017 | WHITE  48.204 | WHITE  47.901 | WHITE  48.951 | BROWN  50.119 | WALL    0.000 |
| WHITE  48.188 | WHITE  47.639 | GREEN  48.238 | WHITE  48.231 | BROWN  48.195 | WALL    0.000 |
| WHITE  47.393 | WHITE  47.039 | WHITE  47.474 | WHITE  48.518 | GREEN  49.704 | WALL    0.000 |
```

Figure 31: Results for Policy Iteration at Iteration 19

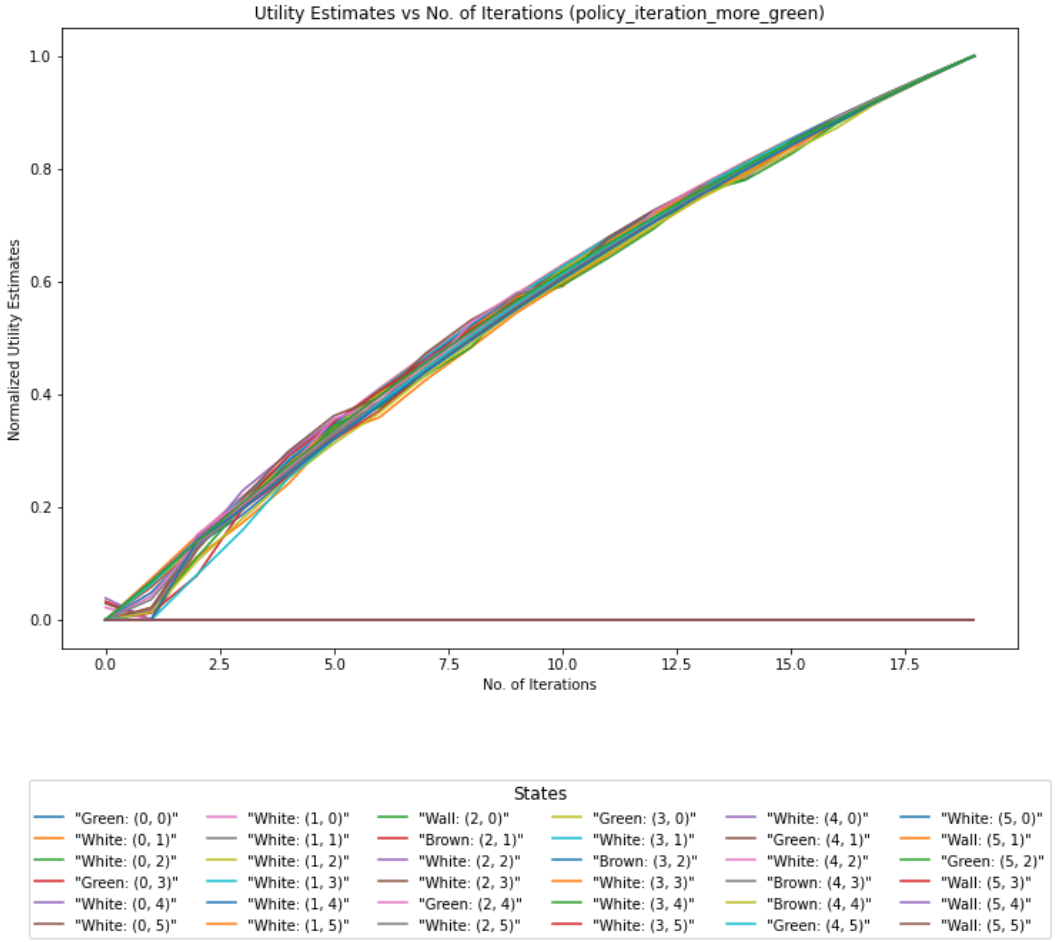

Figure 32: Utility Estimates vs No. of Iterations (policy_iteration_more_green)

4.3. Increasing the Grid Size (Larger Maze)

In the following 7x7 maze (Figure 33), there are one more row and column as compared to the original 6x6 maze.

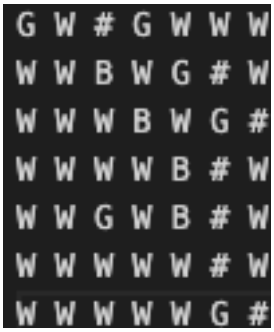

Figure 33: Maze with Increased Grid Size

4.3.1. Results

4.3.1.1. Value Iteration

Figure 34 shows the final plot of the optimal policies and utilities of all states, where it is observed that the number of iterations before convergence remains the same.

```
No. of Iterations: 38
Max Utility Change: 0.683
Plot of Optimal Policy
| GREEN ^ | WHITE < | WALL  - | GREEN ^ | WHITE < | WHITE < | WHITE < |
| WHITE ^ | WHITE < | BROWN > | WHITE ^ | GREEN > | WALL  - | WHITE ^ |
| WHITE ^ | WHITE ^ | WHITE < | BROWN > | WHITE > | GREEN > | WALL  - |
| WHITE ^ | WHITE ^ | WHITE < | WHITE ^ | BROWN ^ | WALL  - | WHITE ^ |
| WHITE ^ | WHITE ^ | GREEN ^ | WHITE ^ | BROWN ^ | WALL  - | WHITE v |
| WHITE ^ | WHITE ^ | WHITE ^ | WHITE > | WHITE v | WALL  - | WHITE v |
| WHITE ^ | WHITE ^ | WHITE > | WHITE > | WHITE > | GREEN > | WALL  - |

Utilities of all States
| GREEN  30.790 | WHITE  30.140 | WALL    0.000 | GREEN  31.246 | WHITE  30.771 | WHITE  30.181 | WHITE  29.529 |
| WHITE  30.140 | WHITE  29.637 | BROWN  28.208 | WHITE  30.581 | GREEN  31.786 | WALL    0.000 | WHITE  28.957 |
| WHITE  29.514 | WHITE  29.121 | WHITE  28.555 | BROWN  29.502 | WHITE  31.813 | GREEN  33.256 | WALL    0.000 |
| WHITE  28.910 | WHITE  28.625 | WHITE  28.264 | WHITE  29.025 | BROWN  29.971 | WALL    0.000 | WHITE -1.297 |
| WHITE  28.339 | WHITE  28.243 | GREEN  28.946 | WHITE  28.551 | BROWN  28.301 | WALL    0.000 | WHITE -1.297 |
| WHITE  27.793 | WHITE  27.833 | WHITE  28.412 | WHITE  28.667 | WHITE  29.726 | WALL    0.000 | WHITE -1.297 |
| WHITE  27.273 | WHITE  27.463 | WHITE  28.537 | WHITE  29.726 | WHITE  31.051 | GREEN  32.427 | WALL    0.000 |
```

Figure 34: Results for Value Iteration at Iteration 38

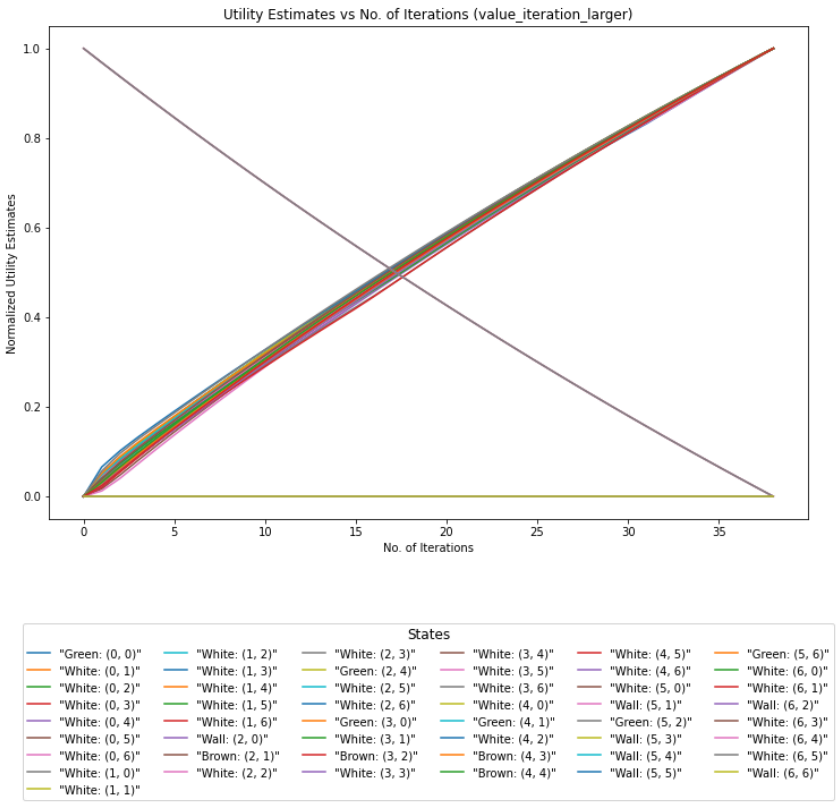

Figure 35: Utility Estimates vs No. of Iterations (value_iteration_larger)

4.3.1.2. Policy Iteration

Figure 36 shows the final plot of the optimal policies and utilities of all states, where it is observed that the number of iterations before convergence for policy iteration increased from 10 iterations to 16 iterations, indicating that the larger size may have increased the number of states and the complexity of the environment. This may make it more difficult for the agent to learn the optimal policy, and hence, slowed down convergence.

```
No. of Iterations: 16
Plot of Optimal Policy
| GREEN ^ | WHITE < | WALL  - | GREEN ^ | WHITE v | WHITE < | WHITE < |
| WHITE ^ | WHITE < | BROWN > | WHITE > | GREEN v | WALL  - | WHITE ^ |
| WHITE ^ | WHITE ^ | WHITE > | BROWN > | WHITE > | GREEN > | WALL  - |
| WHITE ^ | WHITE ^ | WHITE > | WHITE > | BROWN ^ | WALL  - | WHITE ^ |
| WHITE ^ | WHITE > | GREEN v | WHITE v | BROWN v | WALL  - | WHITE ^ |
| WHITE ^ | WHITE > | WHITE > | WHITE v | WHITE v | WALL  - | WHITE ^ |
| WHITE > | WHITE > | WHITE > | WHITE > | WHITE > | GREEN > | WALL  - |

Utilities of all States
| GREEN  44.712 | WHITE  43.908 | WALL    0.000 | GREEN  45.268 | WHITE  44.966 | WHITE  44.227 | WHITE  43.411 |
| WHITE  43.908 | WHITE  43.259 | BROWN  42.498 | WHITE  44.935 | GREEN  46.228 | WALL    0.000 | WHITE  42.695 |
| WHITE  43.136 | WHITE  42.661 | WHITE  42.646 | BROWN  43.918 | WHITE  46.303 | GREEN  47.794 | WALL    0.000 |
| WHITE  42.391 | WHITE  42.055 | WHITE  42.044 | WHITE  43.093 | BROWN  44.269 | WALL    0.000 | WHITE -1.919 |
| WHITE  41.651 | WHITE  41.322 | GREEN  42.394 | WHITE  42.300 | BROWN  42.266 | WALL    0.000 | WHITE -1.939 |
| WHITE  41.000 | WHITE  41.332 | WHITE  42.407 | WHITE  43.473 | WHITE  44.653 | WALL    0.000 | WHITE -1.960 |
| WHITE  40.923 | WHITE  42.073 | WHITE  43.343 | WHITE  44.653 | WHITE  46.012 | GREEN  47.411 | WALL    0.000 |
```

Figure 36: Results for Policy Iteration at Iteration 16

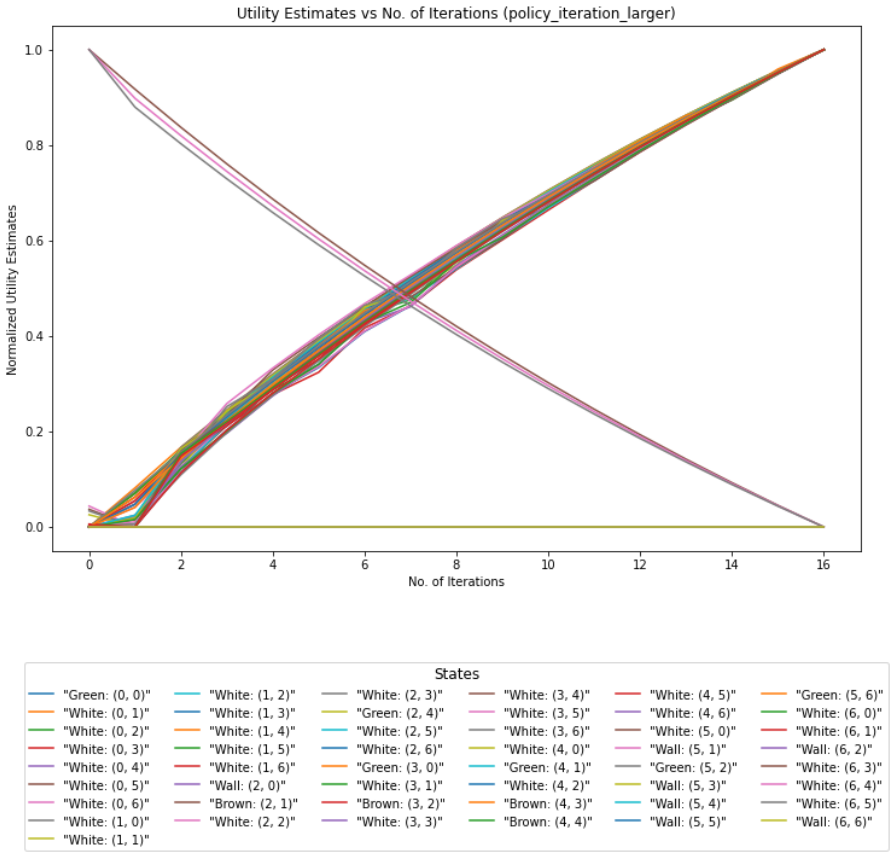

Figure 37: Utility Estimates vs No. of Iterations (policy_iteration_larger)

4.3.2. Observations

With the increase in the size of the maze, the number of states and actions that the agent can take also increases. This may have led to a larger search space and more complex decision-making for the agent. As a result, the optimization algorithm took longer to converge to an optimal policy, and the shape of the line graph became more complex.
The 'x'-shaped line graph may indicate that the optimization algorithm is oscillating between two or more optimal policies as it explores the larger search space.

4.4. Increasing the Complexity of Maze

The following maze (Figure 38) is more complex than the original maze (Figure 22) from Part 1, with more walls and dead ends.

Coordinates of dead ends are as follows:

- (2, 2): Brown cell surrounded by white cells.
- (3, 5): Brown cell surrounded by white cells.
- (4, 5): Brown cell surrounded by white cells.
- (5, 6): Green cell surrounded by white cells.

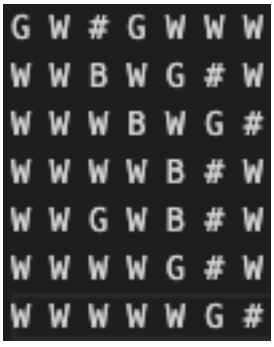

Figure 38: Maze with Increased Complexity

4.4.1. Results

4.4.1.1. Value Iteration

Figure 39 shows the final plot of the optimal policies and utilities of all states, where it is observed that the number of iterations before convergence for value iteration decreased by one.

```
No. of Iterations: 37
Max Utility Change: 0.683
Plot of Optimal Policy
| GREEN ^ | WHITE < | WALL  - | GREEN ^ | WHITE < | WHITE < | WHITE < |
| WHITE ^ | WHITE < | BROWN > | WHITE ^ | GREEN > | WALL  - | WHITE ^ |
| WHITE ^ | WHITE ^ | WHITE < | BROWN > | WHITE > | GREEN > | WALL  - |
| WHITE ^ | WHITE ^ | WHITE < | WHITE ^ | BROWN ^ | WALL  - | WHITE ^ |
| WHITE ^ | WHITE ^ | GREEN v | WHITE v | BROWN v | WALL  - | WHITE v |
| WHITE ^ | WHITE > | WHITE > | WHITE > | GREEN v | WALL  - | WHITE v |
| WHITE ^ | WHITE > | WHITE > | WHITE > | WHITE > | GREEN > | WALL  - |

Utilities of all States
| GREEN  30.156 | WHITE  29.513 | WALL    0.000 | GREEN  30.598 | WHITE  30.129 | WHITE  29.544 | WHITE  28.900 |
| WHITE  29.513 | WHITE  29.016 | BROWN  27.570 | WHITE  29.940 | GREEN  31.132 | WALL    0.000 | WHITE  28.334 |
| WHITE  28.893 | WHITE  28.507 | WHITE  27.947 | BROWN  28.837 | WHITE  31.143 | GREEN  32.582 | WALL    0.000 |
| WHITE  28.296 | WHITE  28.019 | WHITE  27.684 | WHITE  28.375 | BROWN  29.310 | WALL    0.000 | WHITE  -1.270 |
| WHITE  27.733 | WHITE  27.663 | GREEN  28.617 | WHITE  28.596 | BROWN  28.677 | WALL    0.000 | WHITE  -1.270 |
| WHITE  27.211 | WHITE  27.474 | WHITE  28.601 | WHITE  29.760 | GREEN  31.078 | WALL    0.000 | WHITE  -1.270 |
| WHITE  26.760 | WHITE  27.555 | WHITE  28.728 | WHITE  29.924 | WHITE  31.140 | GREEN  32.360 | WALL    0.000 |
```

Figure 39: Results for Value Iteration at Iteration 37

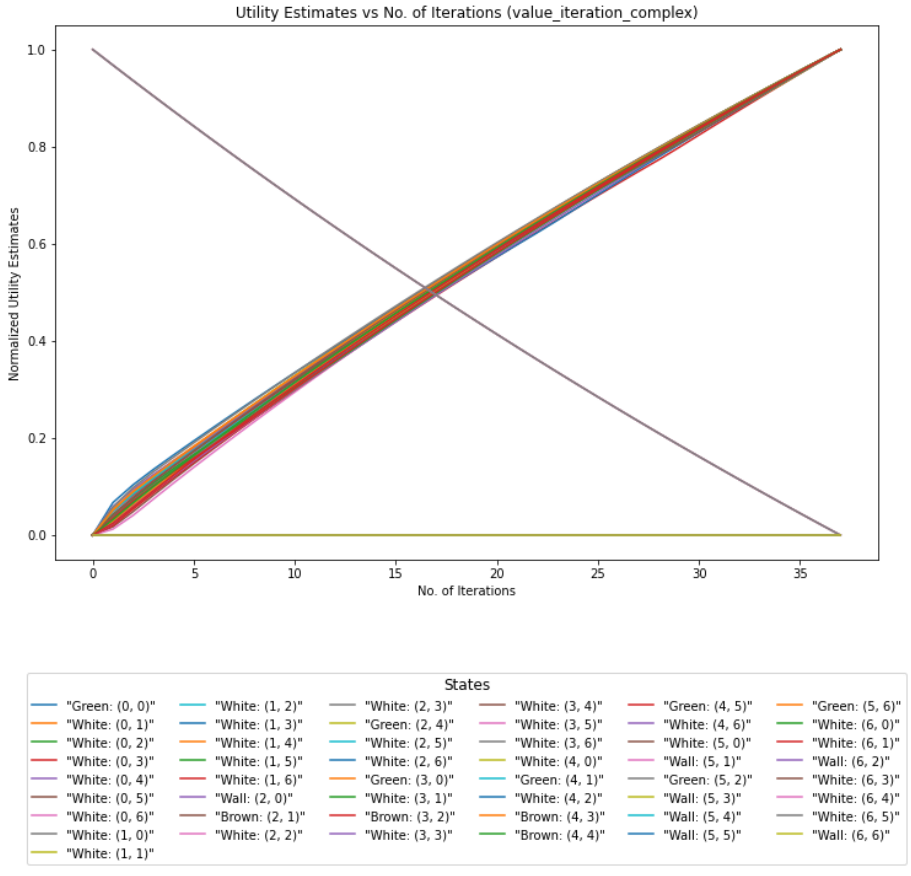

Figure 40: Utility Estimates vs No. of Iterations (value_iteration_complex)

4.4.1.2. Policy Iteration

Figure 41 shows the final plot of the optimal policies and utilities of all states, where it is observed that the number of iterations before convergence for policy iteration increased by 2 iterations. Although the maze is more complex and slowed down convergence, the maze is still relatively small, and hence the agent was able to learn the optimal policy with enough training.

```
No. of Iterations: 12
Plot of Optimal Policy
| GREEN < | WHITE < | WALL  - | GREEN ^ | WHITE v | WHITE < | WHITE < |
| WHITE ^ | WHITE < | BROWN > | WHITE > | GREEN v | WALL  - | WHITE ^ |
| WHITE ^ | WHITE ^ | WHITE > | BROWN > | WHITE > | GREEN > | WALL  - |
| WHITE ^ | WHITE ^ | WHITE v | WHITE > | BROWN ^ | WALL  - | WHITE ^ |
| WHITE ^ | WHITE > | GREEN > | WHITE v | BROWN v | WALL  - | WHITE ^ |
| WHITE ^ | WHITE > | WHITE > | WHITE > | GREEN v | WALL  - | WHITE ^ |
| WHITE > | WHITE > | WHITE > | WHITE > | WHITE > | GREEN > | WALL  - |

Utilities of all States
| GREEN  36.375 | WHITE  35.663 | WALL    0.000 | GREEN  36.274 | WHITE  35.928 | WHITE  35.280 | WHITE  34.565 |
| WHITE  35.663 | WHITE  35.095 | BROWN  33.459 | WHITE  35.887 | GREEN  37.162 | WALL    0.000 | WHITE  33.937 |
| WHITE  34.971 | WHITE  34.490 | WHITE  33.672 | BROWN  34.875 | WHITE  37.223 | GREEN  38.686 | WALL    0.000 |
| WHITE  34.302 | WHITE  33.938 | WHITE  33.600 | WHITE  34.261 | BROWN  35.307 | WALL    0.000 | WHITE -1.556 |
| WHITE  33.664 | WHITE  33.524 | GREEN  34.643 | WHITE  34.644 | BROWN  34.733 | WALL    0.000 | WHITE -1.580 |
| WHITE  33.098 | WHITE  33.496 | WHITE  34.649 | WHITE  35.817 | GREEN  37.143 | WALL    0.000 | WHITE -1.604 |
| WHITE  32.517 | WHITE  33.600 | WHITE  34.784 | WHITE  35.988 | WHITE  37.213 | GREEN  38.442 | WALL    0.000 |
```

Figure 41: Results for Policy Iteration at Iteration 12

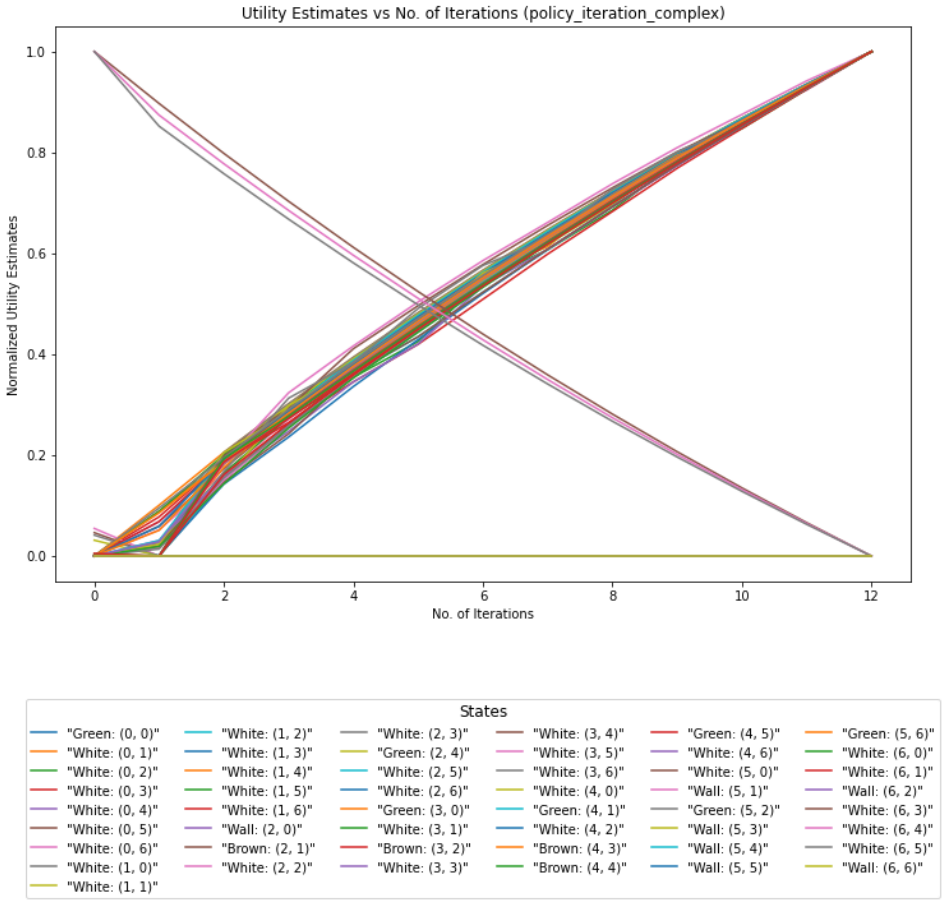

Figure 42: Utility Estimates vs No. of Iterations (policy_iteration_complex)

4.4.2. Observations

An ‘x’-shaped plot is observed with a more complex maze with dead ends, even though the value iteration and policy iterations remain almost the same as the original maze.

Firstly, the dead ends in the maze could be causing the optimization algorithm to have difficulty finding an optimal policy, as dead ends produces a situation where the agent cannot reach the goal state and may get stuck in an infinite loop of unproductive actions. This may lead to oscillations in the optimization algorithm, as it tries to find a way to avoid the dead ends and reach the goal state.

Secondly, the value and policy iterations may be stabilizing quickly due to the structure of the maze. If the structure of the maze is such that there are few possible paths to the goal state, then the optimization algorithm may converge quickly and the line plot may be relatively flat. However, if there are many possible paths to the goal state, then the optimization algorithm may oscillate between different policies, leading to the 'x'-shaped line plot.

5. Conclusion

In conclusion, value iteration and policy iteration are both powerful algorithms for solving Markov Decision Processes (MDPs) and finding optimal policies. Additionally, the number of states and complexity of the environment may have a significant impact on the convergence of these algorithms. As the number of states and the complexity of the environment increase, the convergence time of both algorithms tends to increase as well. In the experiments conducted, it seems to be more evident in policy iteration.

6. References

Jason, N. J. (n.d.). Retrieved from https://github.com/NgoJunHaoJason/CZ4046/tree/master/assignment_1

Junyuan, H. (n.d.). Retrieved from https://github.com/HJunyuan/cz4046-intelligent-agents/tree/master/assignment-1/assignment-1

Norvig., S. R. (2010). *Artificial Intelligence: A Modern Approach.* Prentice-Hall.