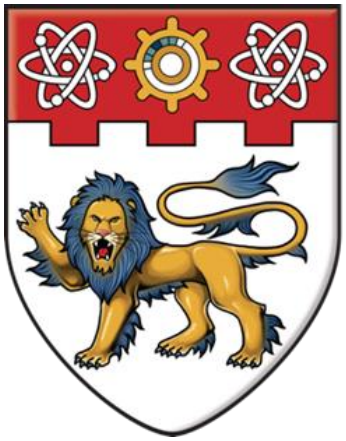


NANYANG TECHNOLOGICAL UNIVERSITY

CZ4046: Intelligent Agents

Assignment 1 Report

Done By: Lee Yih Jie (U1921984F)

School of Computer Science and Engineering

Table of Contents

Contents

1. Overview

1.1. Problem Description

In this assignment, we introduce two algorithms: “Value Iteration” & “Policy Iteration” to obtain an optimal policy for the given grid world environment. The environment is a 6x6 grid world with different tile types: rewards, penalties, walls, and a singular start tile. The transition model states that the agent will have a probability of 0.8 that it moves in its intended direction and a probability of 0.1 each that it moves in a clockwise or an anti-clockwise direction.

	Coordinates used at first for calculation (Row, Col)					
	0	1	2	3	4	5
0	+1 (0,0)	Wall (0,1)	+1 (0,2)	(0,3)	(0,4)	+1 (0,5)
1	(1,0)	-1 (1,1)	(1,2)	+1 (1,3)	Wall (1,4)	-1 (1,5)
2	(2,0)	(2,1)	-1 (2,2)	(2,3)	+1 (2,4)	(2,5)
3	(3,0)	(3,1)	Start (3,2)	-1 (3,3)	(3,4)	+1 (3,5)
4	(4,0)	Wall (4,1)	Wall (4,2)	Wall (4,3)	-1 (4,4)	(4,5)
5	(5,0)	(5,1)	(5,2)	(5,3)	(5,4)	(5,5)

Figure 1: Grid World Environment

1.2. File Organisation

Important files:

- Part_1 folder:
  - Contains all codes related to the first task
    - helper_files folder:
      - Contains all the miscellaneous code implementation which the different policies use.
    - results folder:
      - Contains all the .csv files and plots obtained from running the different policies.
    - value_iteration.py
      - Contains the functions used for the Value Iteration Algorithm
    - policy_iteration.py
      - Contains the functions used for the Policy Iteration Algorithm
- Part_2 folder:
  - Contains all codes related to the second task
    - helper_files folder:
      - Contains all the miscellaneous code implementation which the different policies use.
    - results folder:
      - Contains all the .csv files and plots obtained from running the different policies.
    - value_iteration.py
      - Contains the functions used for the Value Iteration Algorithm
    - policy_iteration.py
      - Contains the functions used for the Policy Iteration Algorithm

2. Task One

Find the optimal policy and the utilities of all the (non-wall) states using both value iteration and policy iteration.

2.1. Value Iteration

2.1.1. Description of implemented solution

The implementation of the Value Iteration is done in Python. It adopts the structure of the Bellman Equation given in (1) below,

$$U_{i+1}(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a) U_i(s') \quad \text{------ (1)}$$

where the Utility of a state *s* in the next iteration U_{i+1} depends on the sum of the reward of s and the discounted max expected utility of the next state s'.

The code for Value Iteration implementation is split into different blocks.

initialise_env(): This is used to create a 6x6 grid world in which there are no actions and the utility values of all states are initialised to zero.

The next few sections are used to implement the equation (1).

next_state_utility() [$U_i(s')$]: This calculates the utility of the next state s' given the current state (row, col) and action taken.

expected_utility() [$\sum_{s'} P(s'|s,a) U_i(s')$]: This calculates the expected utility value of taking action a in state s.

bellman_equation() [$R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a) U_i(s')$]: This implements the full Bellman Equation which returns the maximum utility value of the current state s for the next iteration given the reward and discount factor.

value_iteration(): This implements all previously mentioned sections above and updates the utility values of all states to be used in the next iteration

get_optimal_policy(): This returns the optimal action for each state given the current utility values for each state.

2.1.2. Results of Value Iteration

Optimal Policy

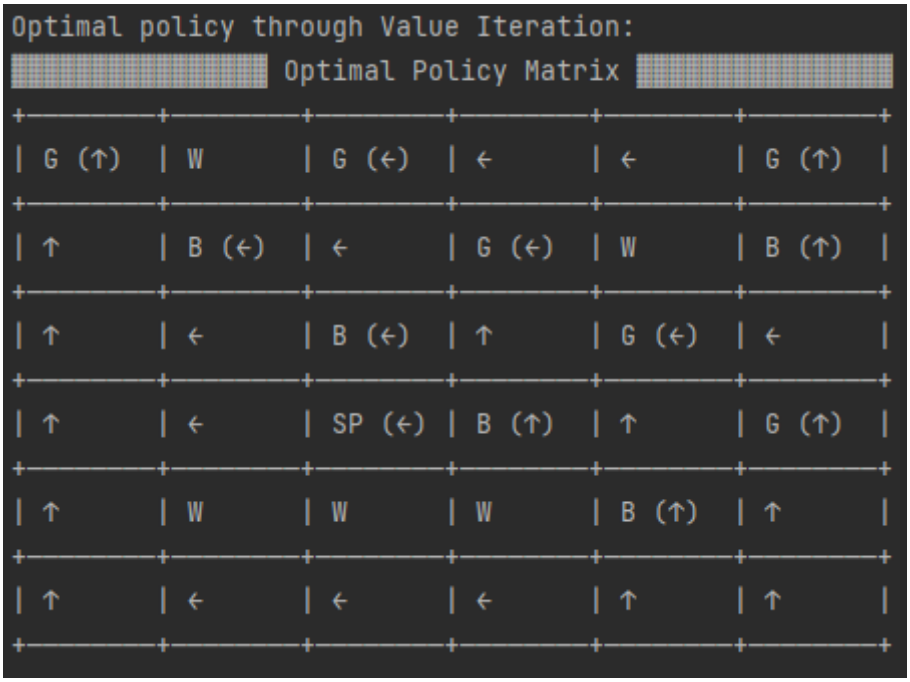


Figure 2: Optimal Policy for Value Iteration

In Figure 2 above, we can observe that the agent has figured out the most optimal sequence of actions which is to reach state (0,0).

Optimal Utility Values

Refer to Appendix A.

Plot of Utility Values as a function of the number of iterations

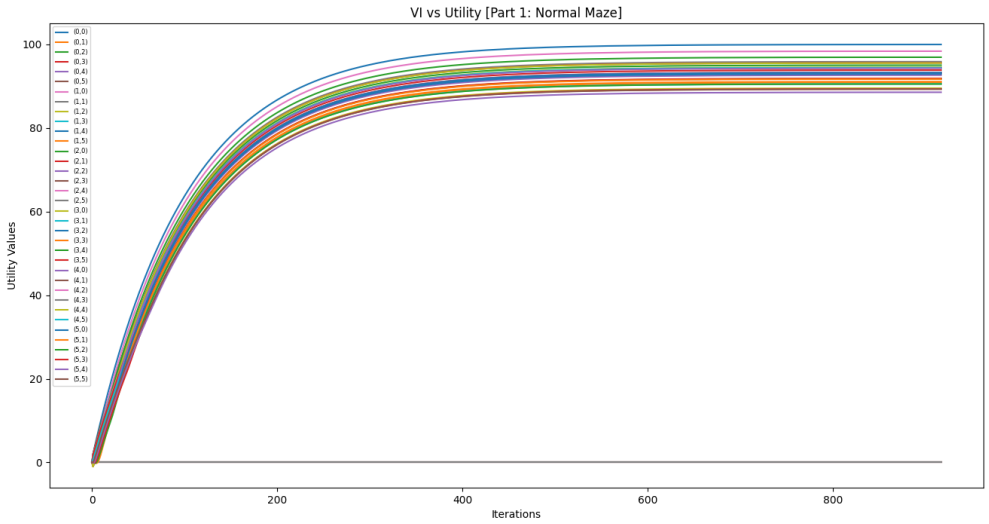


Figure 3: Utility Values vs Number of iterations

A total of 1145 iterations were required to reach convergence using the following values:

Variables	Values
R_MAX	1.00
C	0.10
EPSILON = C * R_MAX	0.10
SMALL_ENOUGH = EPSILON * (1 - DISCOUNT_FACTOR) / DISCOUNT_FACTOR	0.00101010101

Table 1: Values of variables used

2.1.3. Value Iteration Conclusion

In this implementation, a total of 1145 iterations were required to reach convergence. Experimentation done with different values of C and R_MAX variables yielded no significant change in the optimal policy found. The only difference was the number of iterations required for convergence, which increased when EPSILON was increased or and decreased when EPSILON was decreased. From figure 3 above, we can observe that the relative utility values of the different states remain the same after about 400 iterations, which could imply that the optimal policy could have been found much earlier if the convergence value SMALL_ENOUGH was changed more significantly

2.2. Policy Iteration

2.2.1. Description of implemented solution

The code for Policy Iteration implementation is again split into different blocks:

initialise_env(): This is used to create a 6x6 grid world in which the utility values of all states are initialised to zero and a random policy is generated for each state, in which each state has a randomly generated action ('UP', 'DOWN', 'LEFT', 'RIGHT').

next_state_utility() [$U_i(s')$]: This calculates the utility of the next state s' given the current state (row, col) and action taken.

expected_utility() [$\sum_{s'} P(s'|s,a)U_i(s')$]: This calculates the expected utility value of taking action a in state s.

bellman_equation(): This implements the modified Bellman Equation in which the utility is returned for the specific policy for the current state.

policy_evaluation(): This performs evaluation on the current policy for all states

policy_iteration(): This implements all previously mentioned sections above and updates the currently policy based on compared the utility values of the policy in the i+1 iteration and i iteration.

2.2.2. Results of Policy Iteration

Optimal Policy

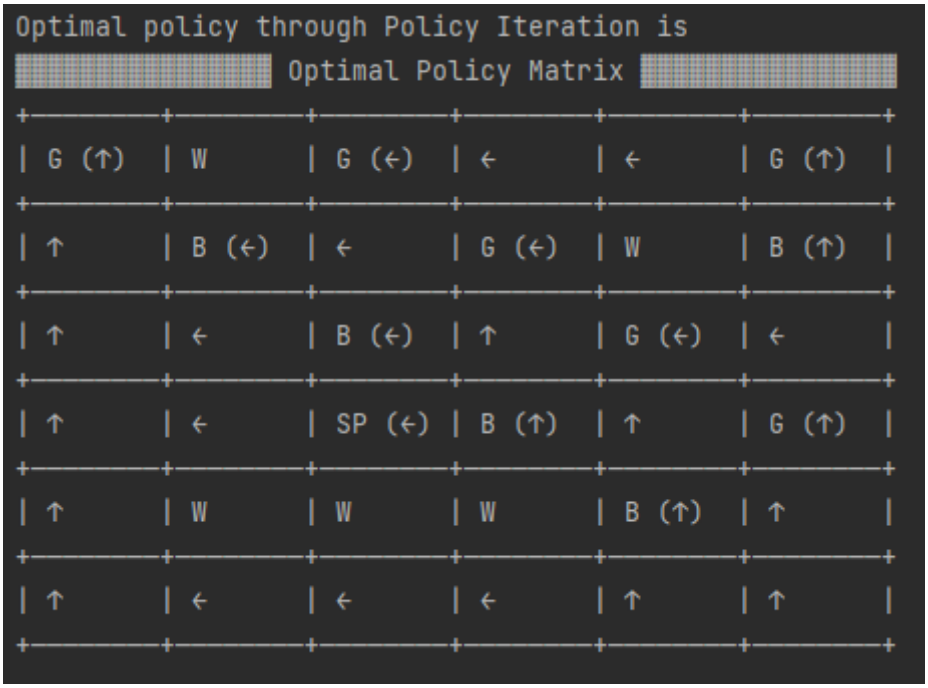


Figure 4: Optimal Policy for Policy Iteration

Again, in Figure 4 above, we can observe that the agent has figured out the most optimal sequence of actions which is to reach state (0,0).

Optimal Utility Values

Refer to Appendix B.

Plot of Utility Values as a function of the number of iterations

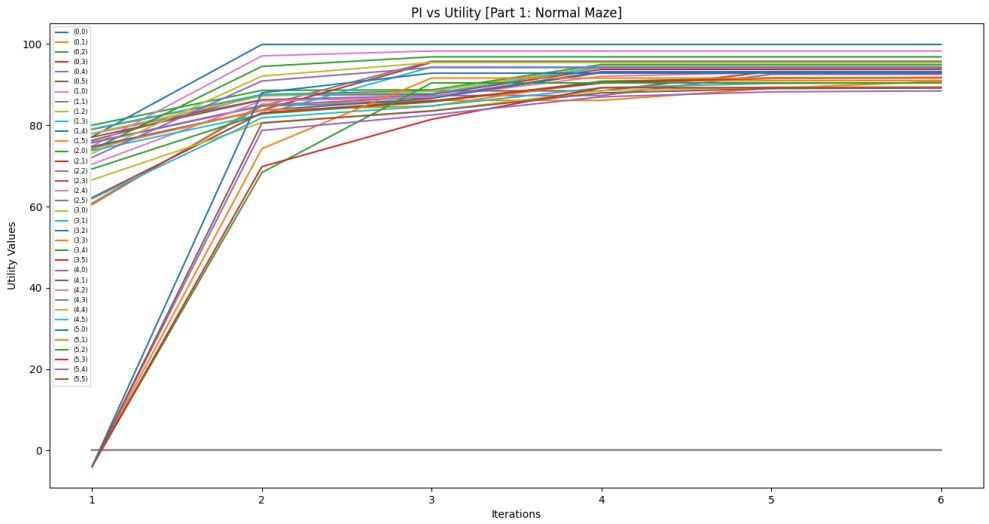


Figure 5: Utility Values vs Number of iterations

2.2.3. Policy Iteration Conclusion

In this implementation, a total of 6 iterations were required to reach convergence. Experimentation with the K parameter yielded no significant change in results. We can observe that using Policy Iteration finished in significantly lesser iterations as compared to using Value Iteration.

2.3. Conclusion

Comparing both algorithms, we observe that that Value Iteration finishes its iterations and converges much faster than Policy Iteration despite having a much larger number of iterations. This could be explained by the fact that Value Iteration calculates the optimal policy by first calculating the maximum utility value of each action in each state. This implies that Value Iteration would converge more quickly if the number of actions to be taken is small.

We can also observe that policy iteration takes different numbers of iterations to converge after running it multiple times (5/6/7). This is because the policy is instantiated with randomly selected actions before Policy Iteration is done and there is a possibility that some sequences of actions randomly selected already had high utility values.

In conclusion, give the current task and its parameters, we deduce that Policy Iteration seems to be the better choice to use as compared to Value Iteration.

3. Task 2

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?

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

3.1. Description of implemented experiments

To tackle this part of the problem on how the number of states as well as the complexity can affect convergence, a more complex maze was implemented.

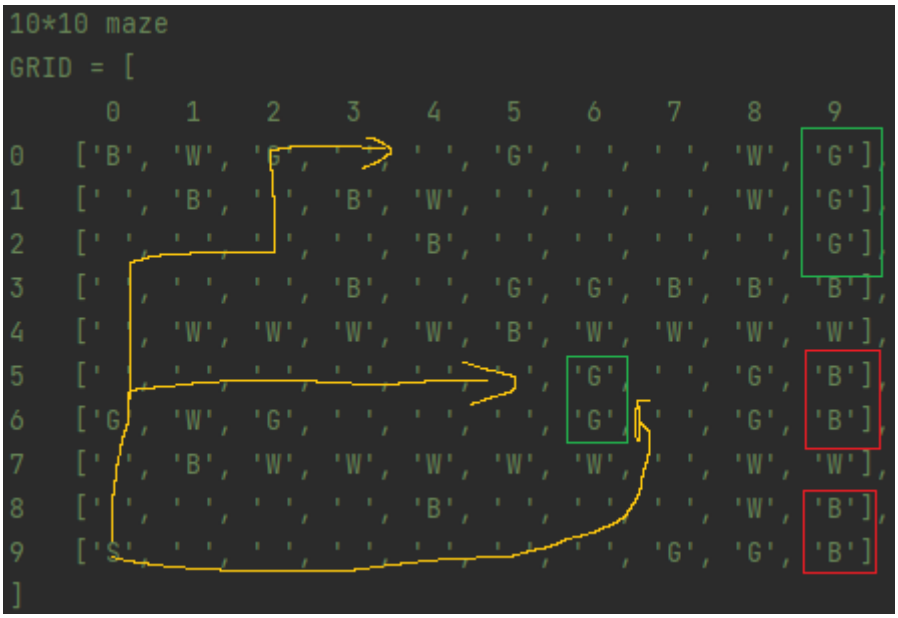


Figure 6: Complicated Grid

The above grid world is now more complex than the one give in Task 1 due to:

- Now a 10x10 grid instead of a 6x6 grid, hence, a greater number of available states for the agent to reach.
- “Negative” ends outlined in red in which positive rewards states (G) are near it but reward of states (B) is negative.
- The yellow arrows are the different possible paths that the agent could take to reach groups of states with positive rewards, with each path having negative rewards in the way to influence the route of the agent

3.2. Results

3.2.1. Value Iteration

Optimal Policy

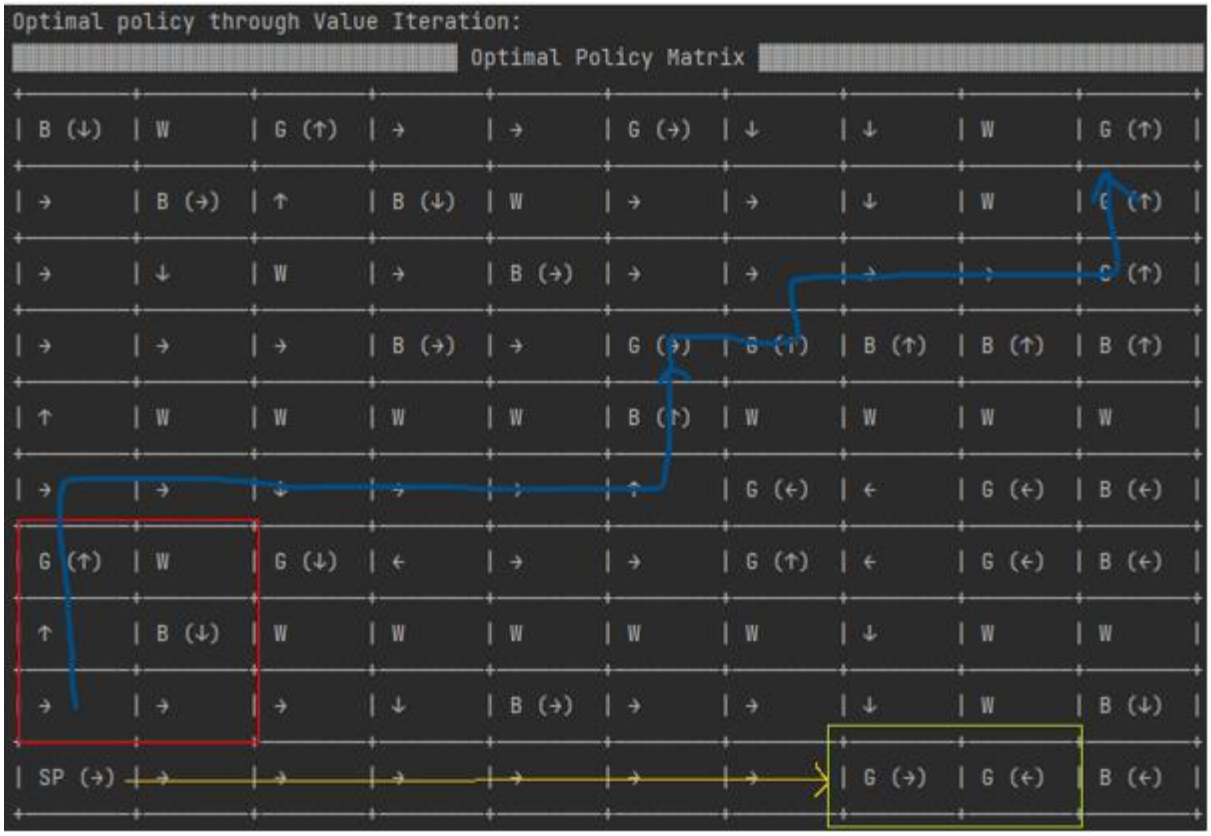


Figure 7: Optimal Policy

We can see that the optimal path for the agent is only the one outlined in yellow towards the positive reward states outlined in green. For the area outlined in red, and the route the agent could have taken in blue, the algorithm determined that the negative reward state there affected the utility too much that coupled with the increased distance away from the start point for the other positive reward clusters, the algorithm choose the nearest positive reward cluster instead

Final Utility Values

Refer to Appendix C

Plot of Utility Values as a function of the number of iterations

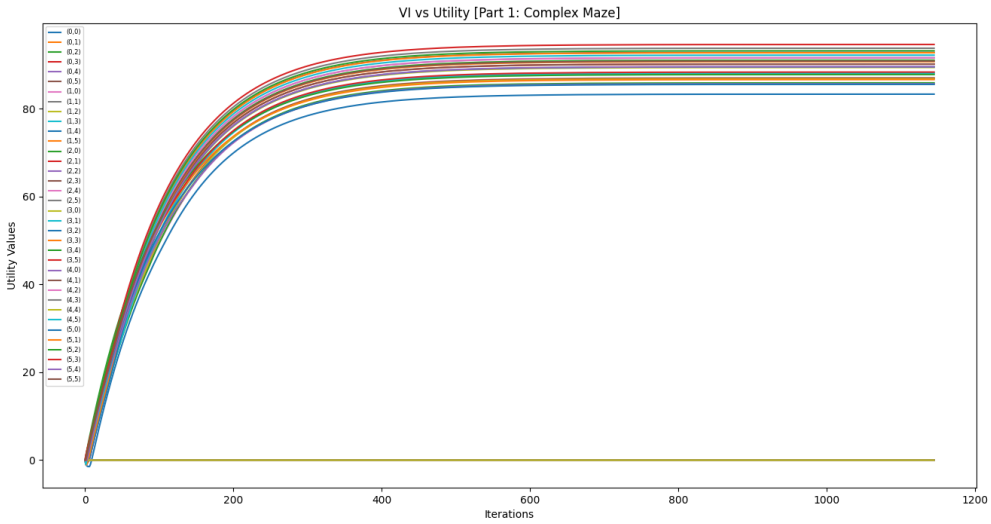


Figure 8: Utility Values vs Number of iterations

3.2.2. Policy Iteration

Optimal Policy

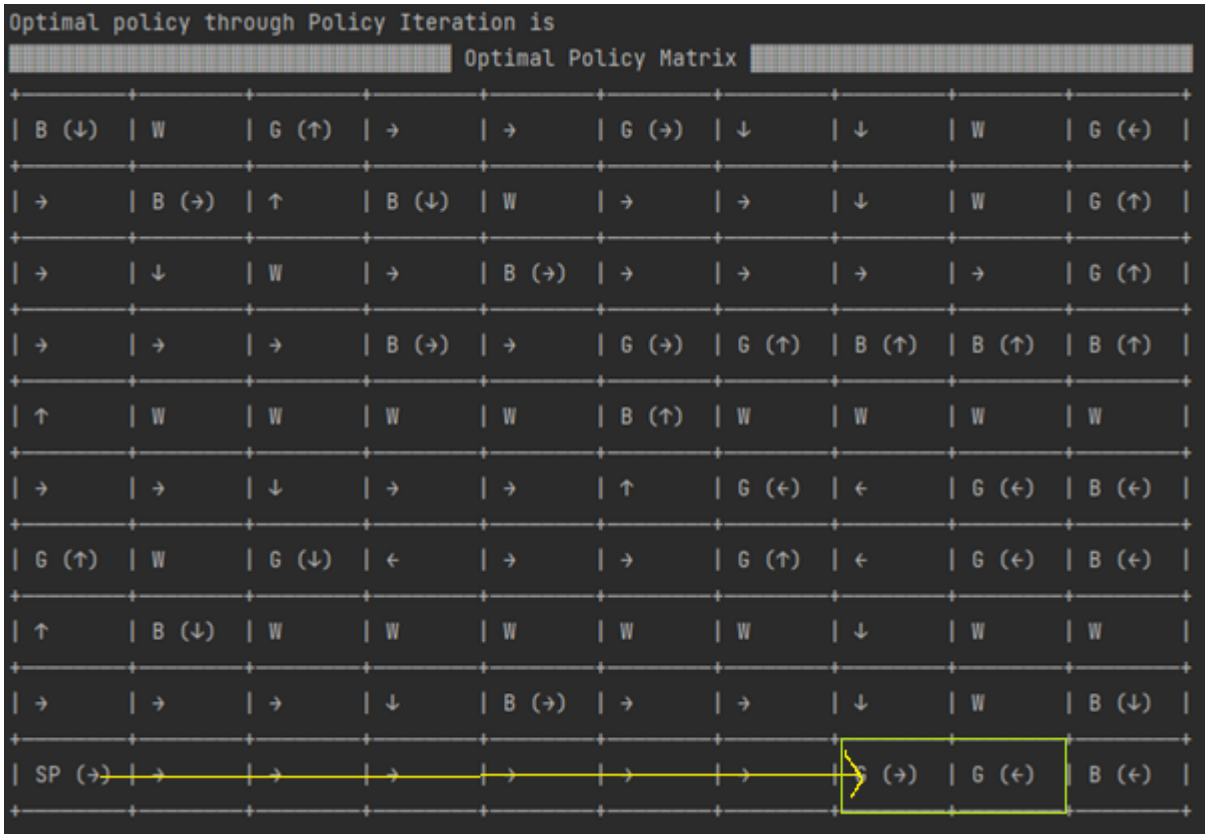


Figure 9: Optimal Policy

We can observe that using Policy Iteration does not change the optimal policy, where the algorithm still chooses the nearest cluster of positive rewards instead of the ones a greater distance away.

Final Utility Values

Refer to Appendix D

Plot of Utility Values as a function of the number of iterations

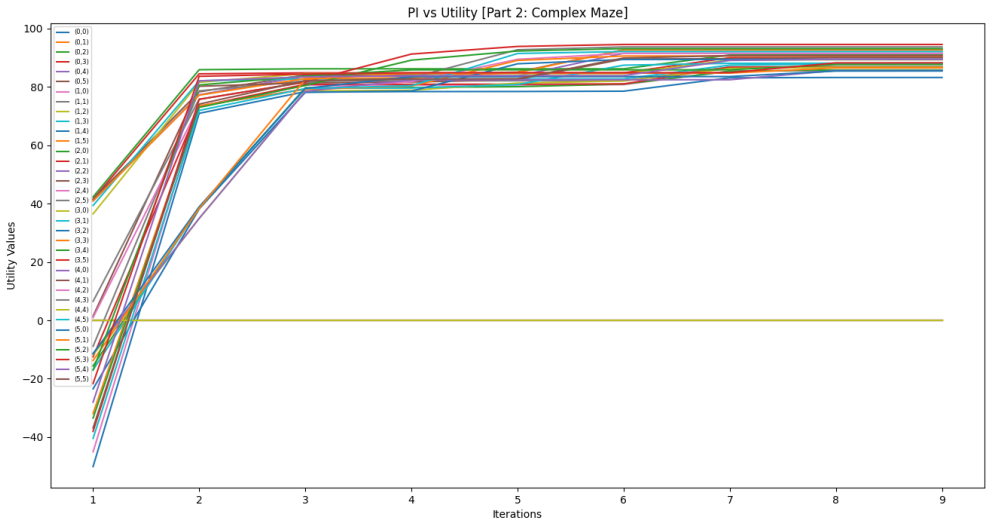


Figure 10: Utility Values vs Number of iterations

3.2.3. Increasing Discount factor

Since the both algorithms seem to favour choosing the shortest path over the greater rewards, we decided to increase the value of the discount factor from 0.99 to 0.999 so that the agent will try to prioritise long term rewards instead.

Value Iteration

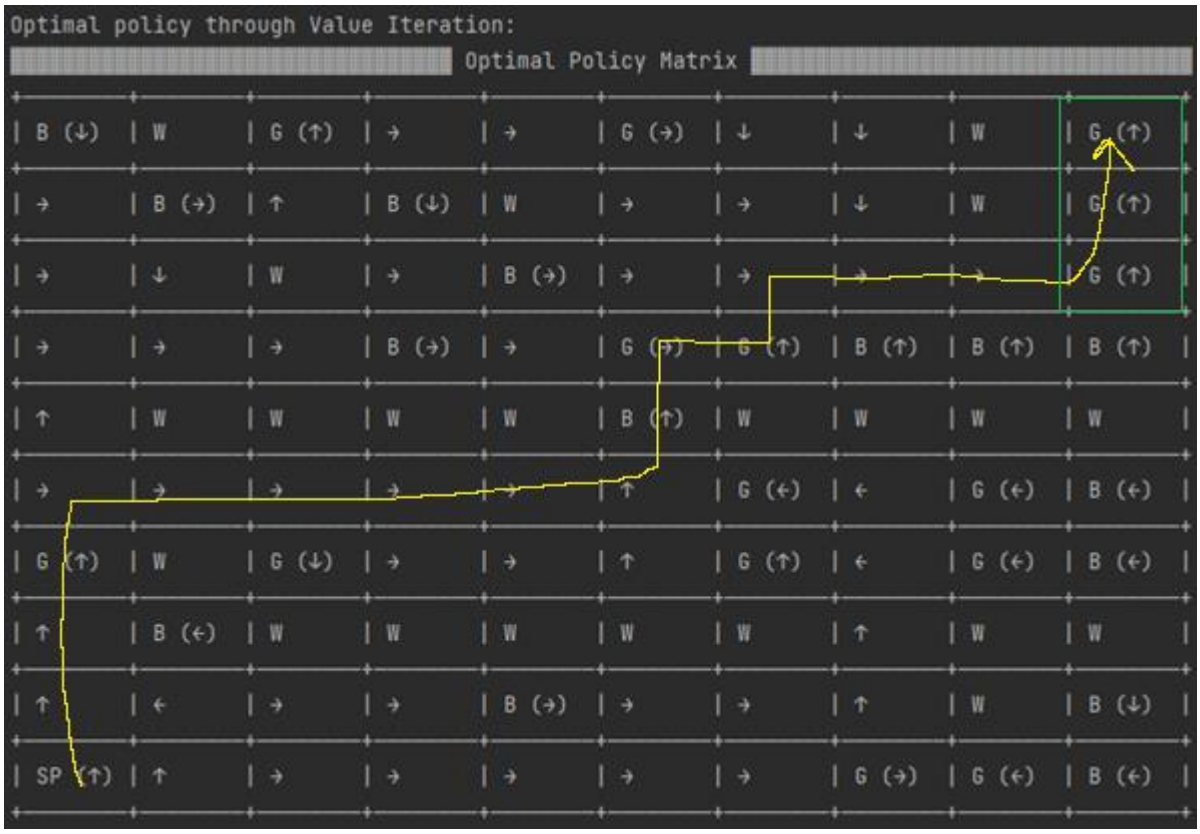


Figure 11: Optimal Policy

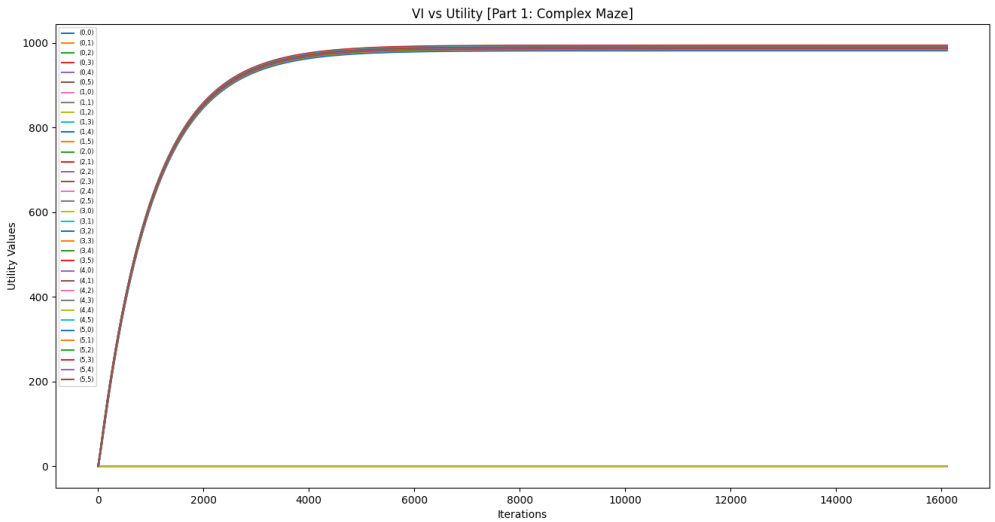


Figure 12: Utility Values vs Number of iterations

We can see that by increasing the discount factor for Value Iteration, the agent prioritised long-term value states rather than short-term ones and choose to travel to the greater rewards states further away, hence achieving the optimal policy with almost maximum utility values. This comes at a

drawback of taking almost 10 times the number of iterations to converge (1145 vs 16110). Refer to Appendix E for the Final utility Values.

Policy Iteration

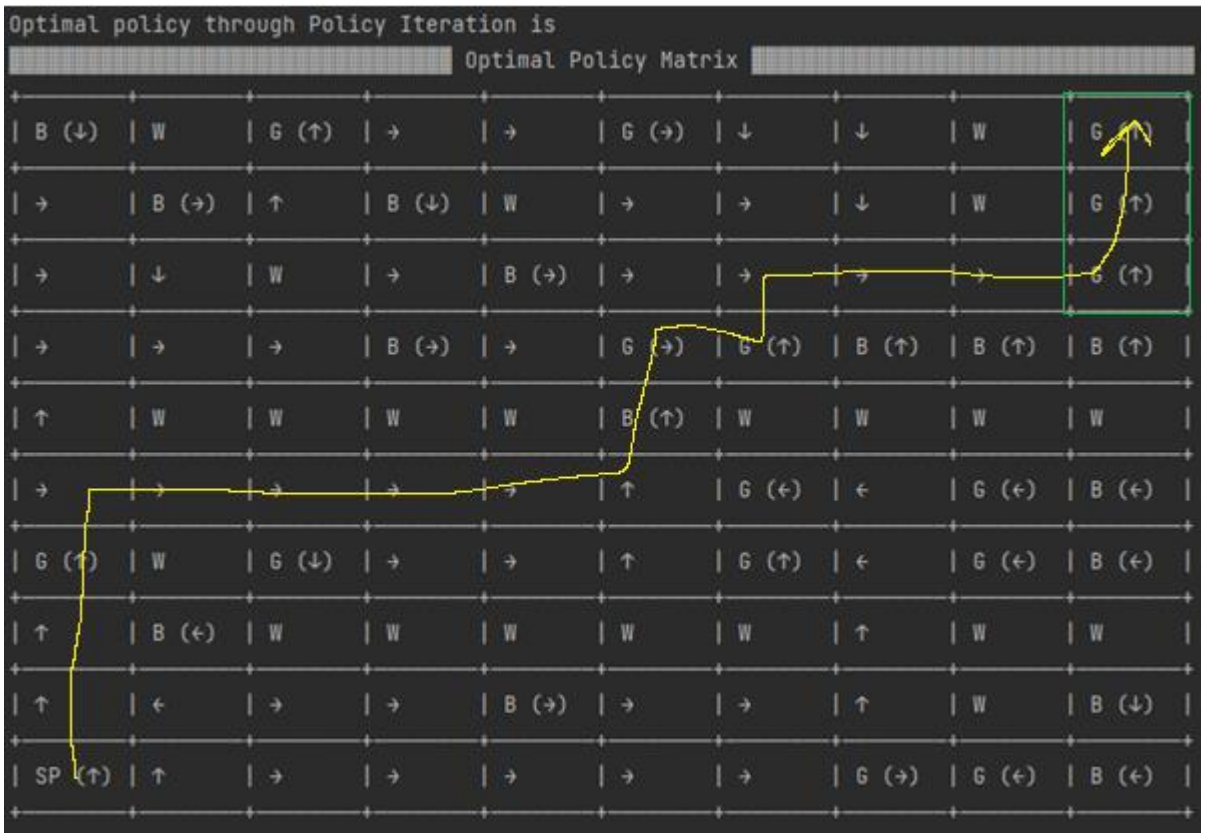


Figure 13: Optimal Policy

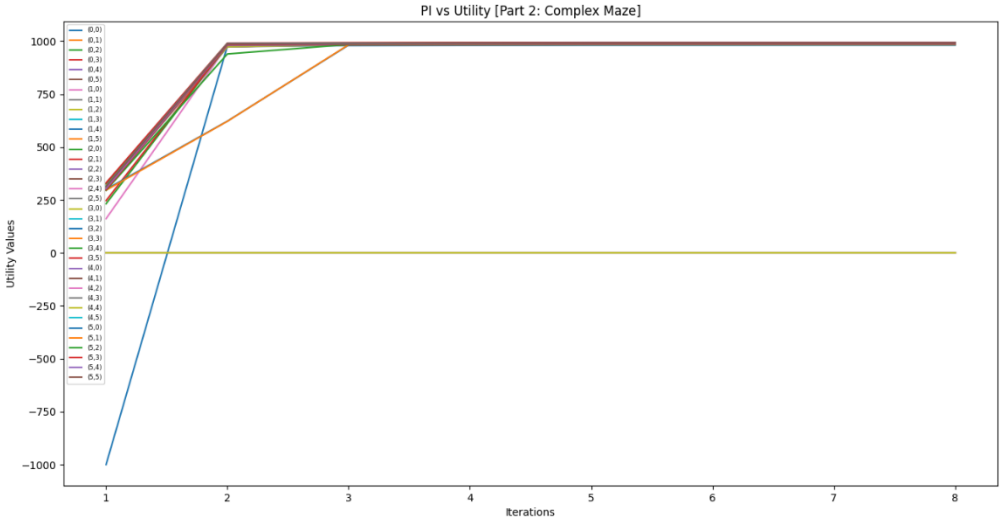


Figure 14: Utility Values vs Number of iterations

Likewise, we can observe that increasing the discount factor for Policy Iteration also allowed the agent to prioritise the long-term reward states and is able to achieve the same optimal policy as Value Iteration. Again, this comes at a drawback of each iteration of the Policy Iteration taking a much longer time to execute.

3.3. Conclusion

We can conclude that the complexity and size of the maze increases the number of iterations for both algorithms to converge, as well as the overall execution time for each iteration. Furthermore, we can also observe that increasing the discount factor allows the agent to prioritise further greater positive rewards, albeit at a very much longer execution time. Hence, it is worthy to investigate a optimized discount factor value that allows the agent to get the optimal policy without taking such a long duration to execute its iterations.

Appendix A

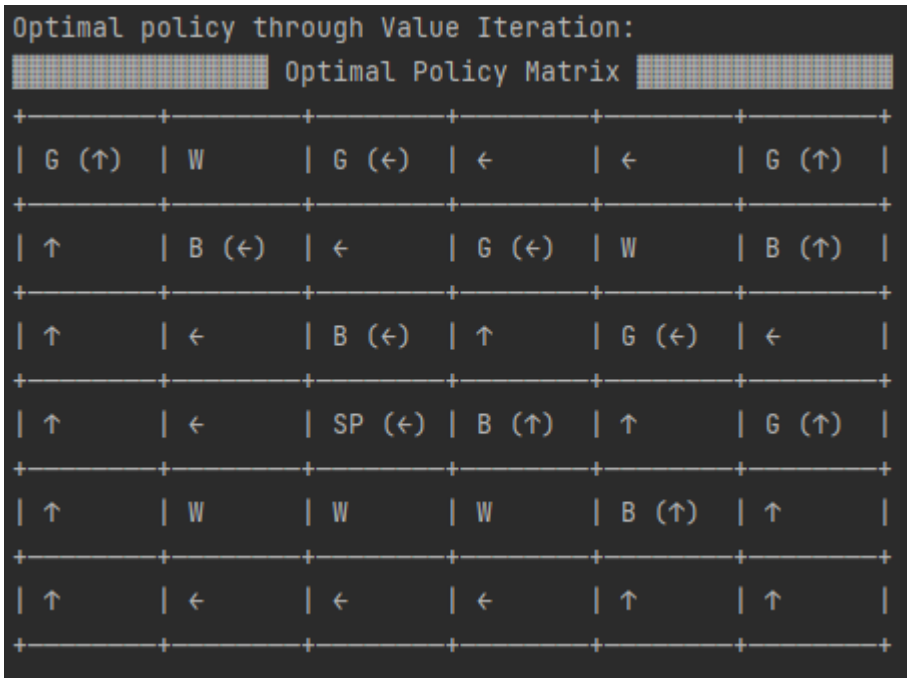
Initial Utility Values

```
Displaying initial policy:
                    Grid World Matrix
+---------+---------+---------+---------+---------+---------+
|G 0      | W 0     |G 0      |  0      |  0      |G 0      |
+---------+---------+---------+---------+---------+---------+
|  0      |B 0      |  0      |G 0      | W 0     |B 0      |
+---------+---------+---------+---------+---------+---------+
|  0      |  0      |B 0      |  0      |G 0      |  0      |
+---------+---------+---------+---------+---------+---------+
|  0      |  0      |  0      |B 0      |  0      |G 0      |
+---------+---------+---------+---------+---------+---------+
|  0      | W 0     | W 0     | W 0     |B 0      |  0      |
+---------+---------+---------+---------+---------+---------+
|  0      |  0      |  0      |  0      |  0      |  0      |
+---------+---------+---------+---------+---------+---------+
```

Final Utility Values

```
Iteration Number: 1145
                    Grid World Matrix
+---------+---------+---------+---------+---------+---------+
|G 99.99  | W 0.0   |G 95.04  |  93.87  |  92.65  |G 93.32  |
+---------+---------+---------+---------+---------+---------+
|  98.39  |B 95.88  |  94.54  |G 94.39  | W 0.0   |B 90.91  |
+---------+---------+---------+---------+---------+---------+
|  96.94  |  95.58  |B 93.29  |  93.17  |G 93.10  |  91.79  |
+---------+---------+---------+---------+---------+---------+
|  95.55  |  94.45  |  93.23  |B 91.11  |  91.81  |G 91.88  |
+---------+---------+---------+---------+---------+---------+
|  94.31  | W 0.0   | W 0.0   | W 0.0   |B 89.54  |  90.56  |
+---------+---------+---------+---------+---------+---------+
|  92.93  |  91.72  |  90.53  |  89.35  |  88.56  |  89.29  |
+---------+---------+---------+---------+---------+---------+
```

Optimal Policy



Appendix B

Initial Policy and Utility Values

Displaying initial policy

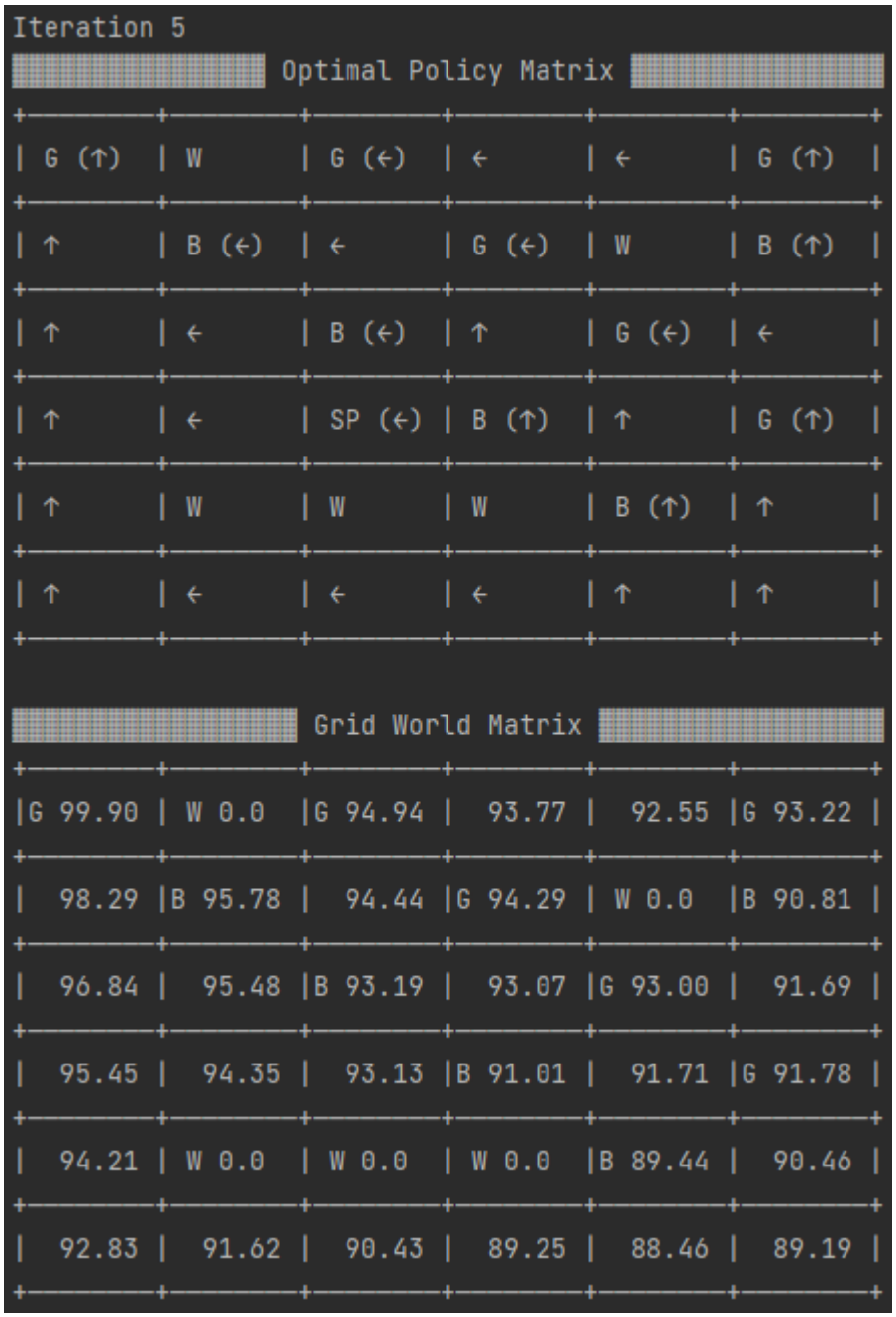
Optimal Policy Matrix

G (←)	W	G (←)	←	←	G (←)
→	B (→)	→	G (→)	W	B (→)
↑	↑	B (↑)	↑	G (↑)	↑
↑	↑	SP (↑)	B (↑)	↑	G (↑)
↑	W	W	W	B (↑)	↑
↓	↓	↓	↓	↓	↓

Grid World Matrix

G 77.15	W 0.0	G 80.03	78.99	77.95	G 76.27
74.86	B 75.68	78.02	G 79.08	W 0.0	B 60.44
73.97	74.73	B 75.82	77.07	G 70.43	60.82
73.08	73.77	74.66	B 74.28	69.25	G 62.18
72.12	W 0.0	W 0.0	W 0.0	B 66.57	61.92
-3.99	-3.99	-3.99	-3.99	-3.99	-3.99

Final Policy and Utility Values (Iteration 5)



Iteration 5

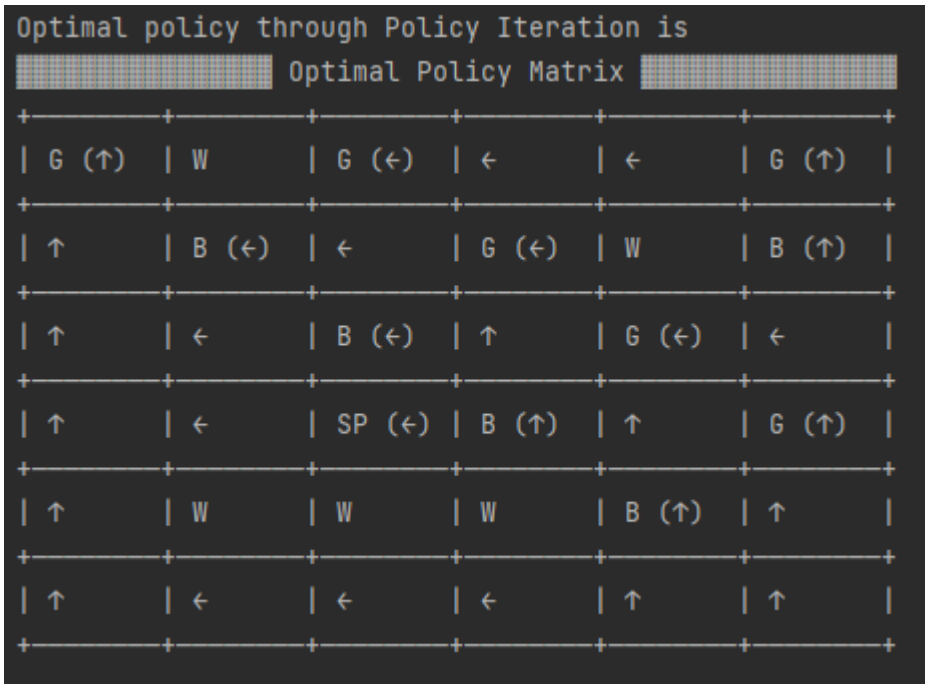
Optimal Policy Matrix

G (↑)	W	G (←)	←	←	G (↑)
↑	B (←)	←	G (←)	W	B (↑)
↑	←	B (←)	↑	G (←)	←
↑	←	SP (←)	B (↑)	↑	G (↑)
↑	W	W	W	B (↑)	↑
↑	←	←	←	↑	↑

Grid World Matrix

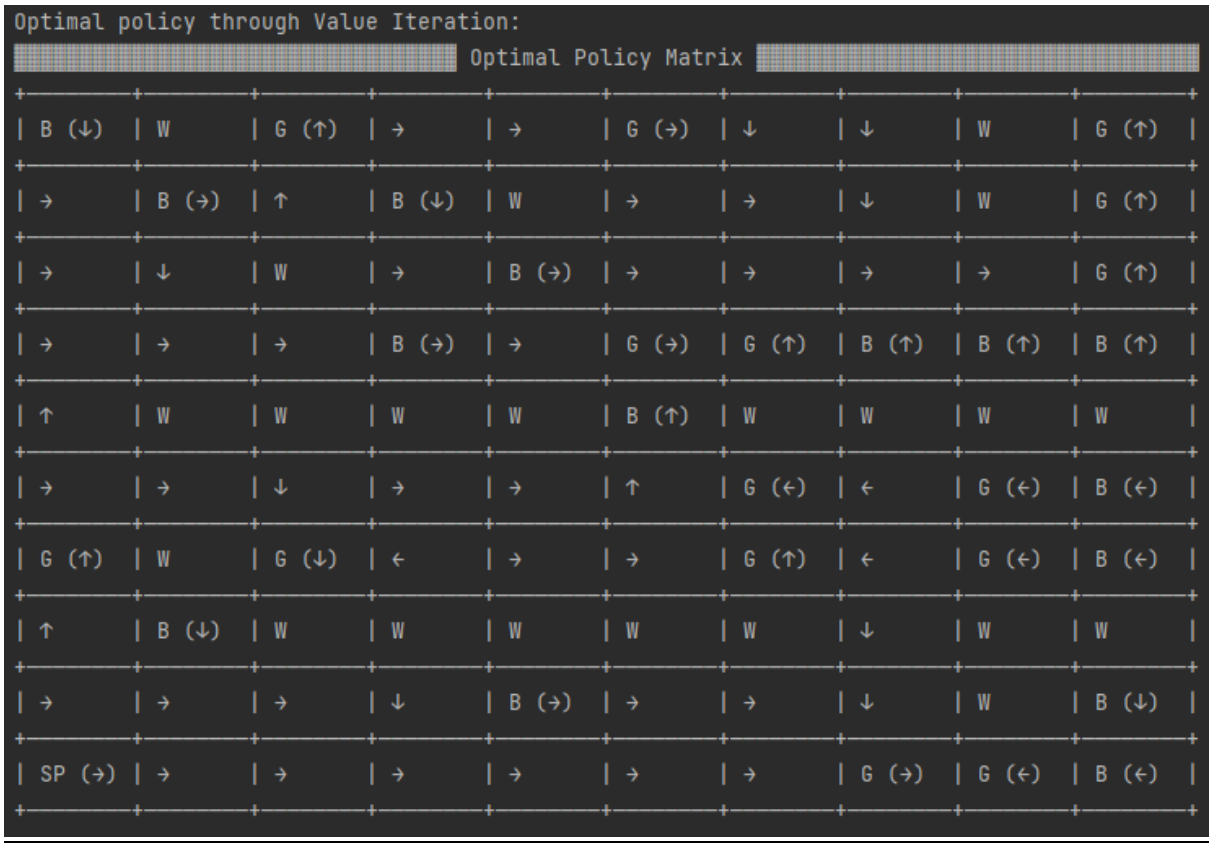
G 99.90	W 0.0	G 94.94	93.77	92.55	G 93.22
98.29	B 95.78	94.44	G 94.29	W 0.0	B 90.81
96.84	95.48	B 93.19	93.07	G 93.00	91.69
95.45	94.35	93.13	B 91.01	91.71	G 91.78
94.21	W 0.0	W 0.0	W 0.0	B 89.44	90.46
92.83	91.62	90.43	89.25	88.46	89.19

Optimal Policy

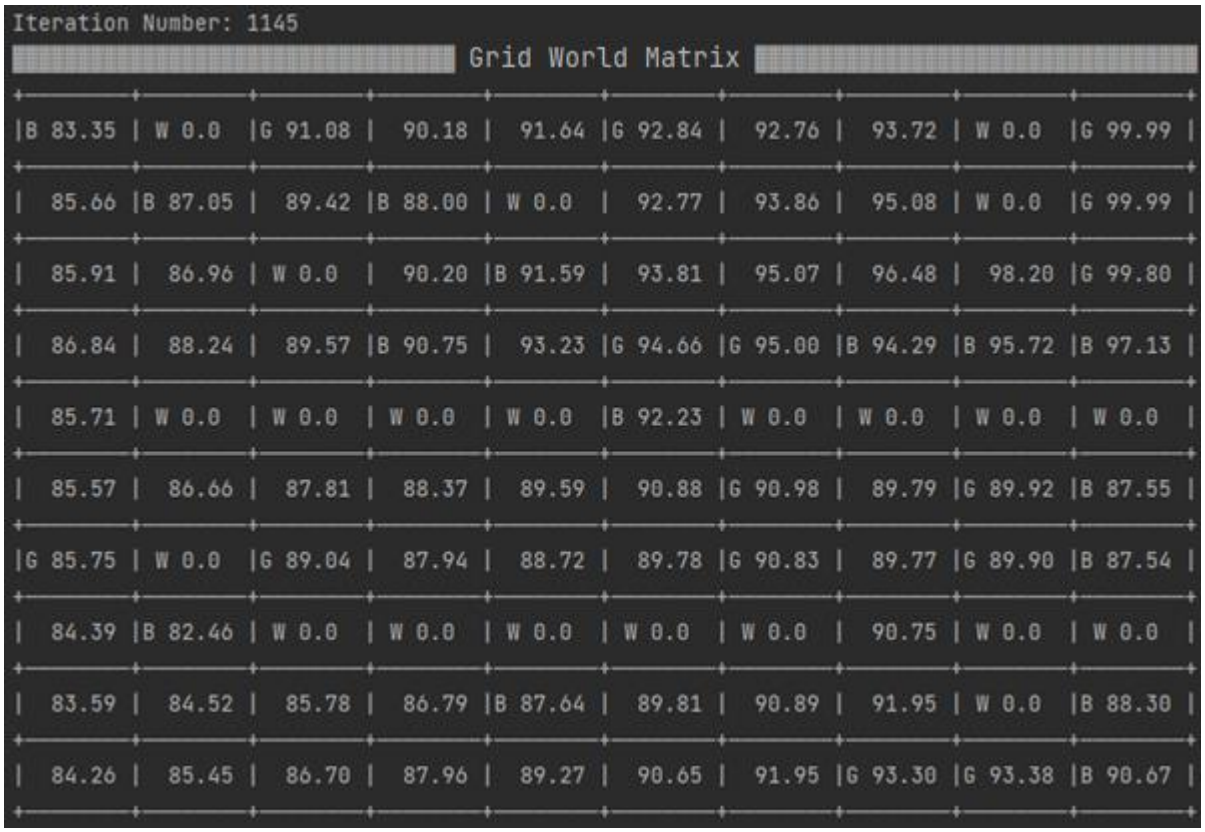


Appendix C

Optimal Policy



Final Iteration (1145)



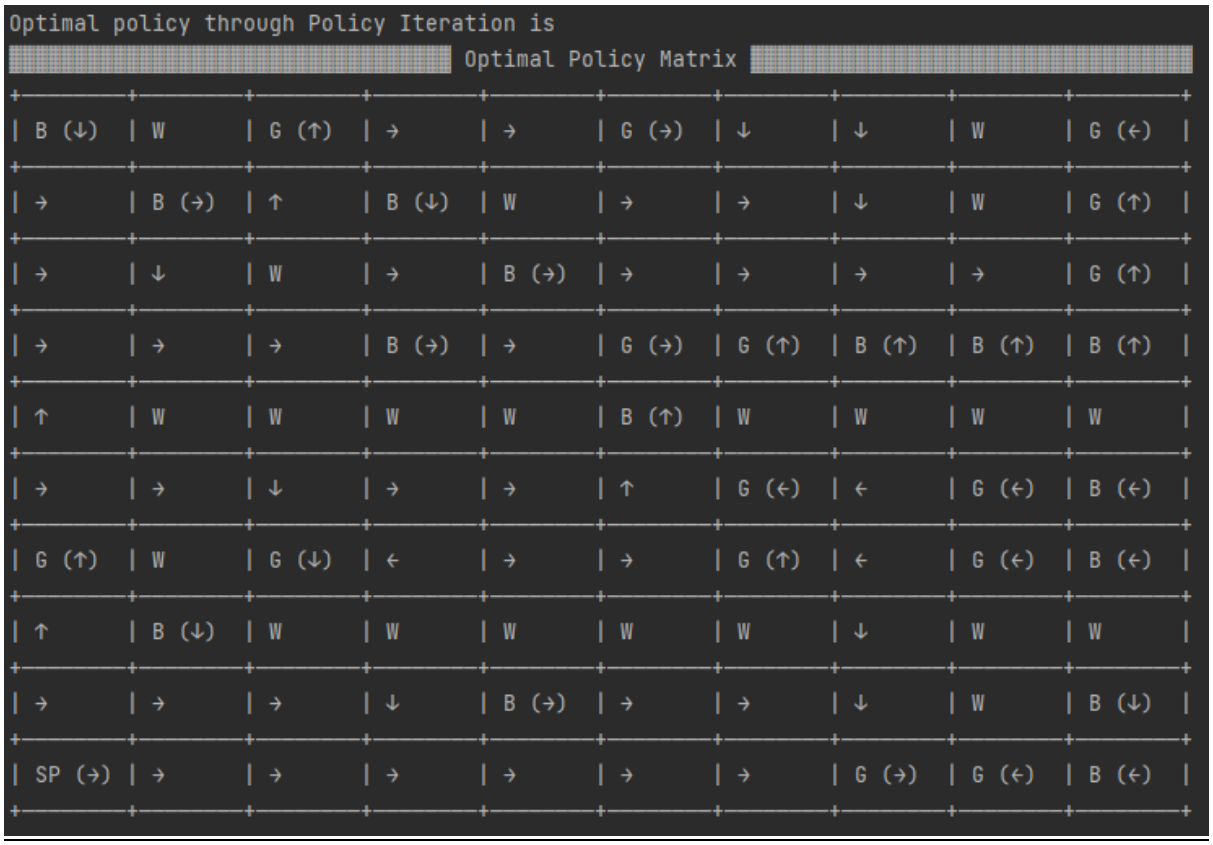
Iteration Number: 1145

Grid World Matrix

B 83.35	W 0.0	G 91.08	90.18	91.64	G 92.84	92.76	93.72	W 0.0	G 99.99
85.66	B 87.05	89.42	B 88.00	W 0.0	92.77	93.86	95.08	W 0.0	G 99.99
85.91	86.96	W 0.0	90.20	B 91.59	93.81	95.07	96.48	98.20	G 99.80
86.84	88.24	89.57	B 90.75	93.23	G 94.66	G 95.00	B 94.29	B 95.72	B 97.13
85.71	W 0.0	W 0.0	W 0.0	W 0.0	B 92.23	W 0.0	W 0.0	W 0.0	W 0.0
85.57	86.66	87.81	88.37	89.59	90.88	G 90.98	89.79	G 89.92	B 87.55
G 85.75	W 0.0	G 89.04	87.94	88.72	89.78	G 90.83	89.77	G 89.90	B 87.54
84.39	B 82.46	W 0.0	W 0.0	W 0.0	W 0.0	W 0.0	90.75	W 0.0	W 0.0
83.59	84.52	85.78	86.79	B 87.64	89.81	90.89	91.95	W 0.0	B 88.30
84.26	85.45	86.70	87.96	89.27	90.65	91.95	G 93.30	G 93.38	B 90.67

Appendix D

Optimal Policy



Final Policy and Utility Values (8)

Iteration 8

Optimal Policy Matrix

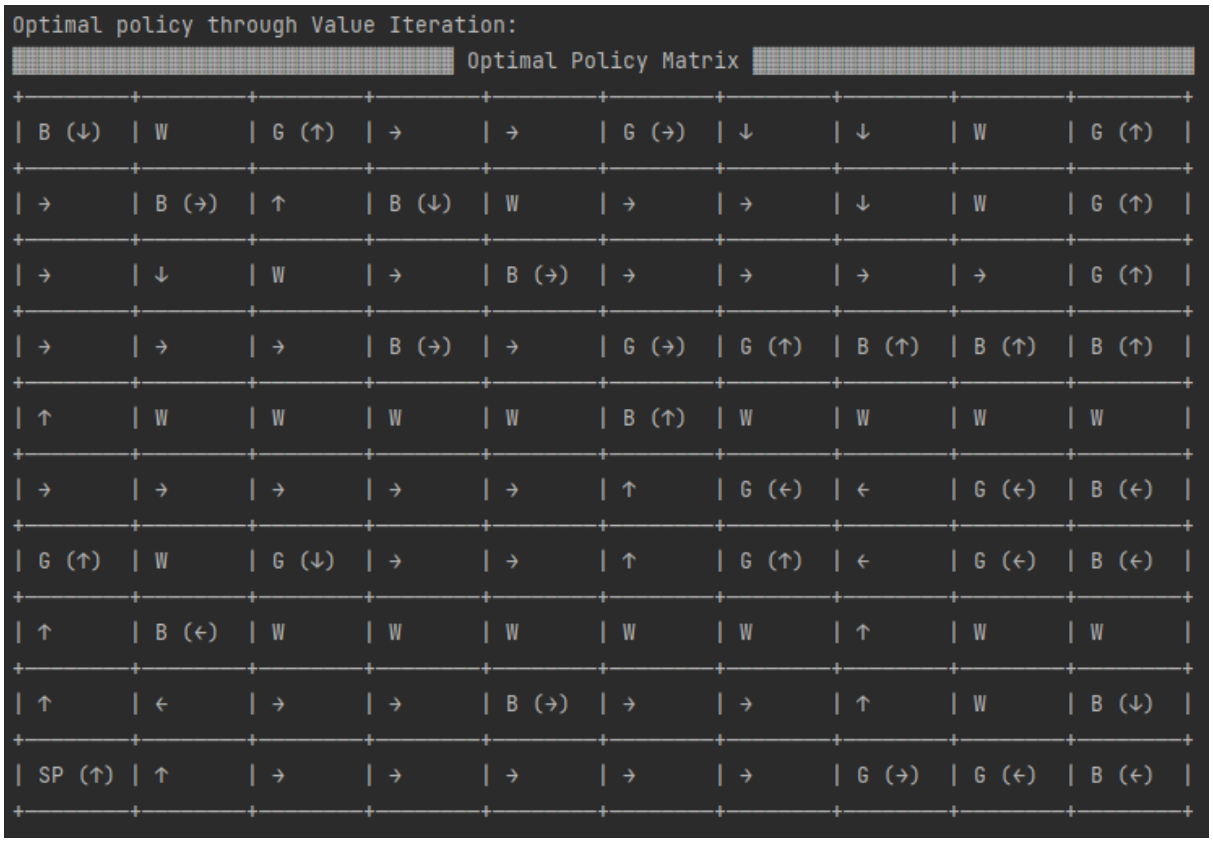
B (↓)	W	G (↑)	→	→	G (→)	↓	↓	W	G (←)
→	B (→)	↑	B (↓)	W	→	→	↓	W	G (↑)
→	↓	W	→	B (→)	→	→	→	→	G (↑)
→	→	→	B (→)	→	G (→)	G (↑)	B (↑)	B (↑)	B (↑)
↑	W	W	W	W	B (↑)	W	W	W	W
→	→	↓	→	→	↑	G (←)	←	G (←)	B (←)
G (↑)	W	G (↓)	←	→	→	G (↑)	←	G (←)	B (←)
↑	B (↓)	W	W	W	W	W	↓	W	W
→	→	→	↓	B (→)	→	→	↓	W	B (↓)
SP (→)	→	→	→	→	→	→	G (→)	G (←)	B (←)

Grid World Matrix

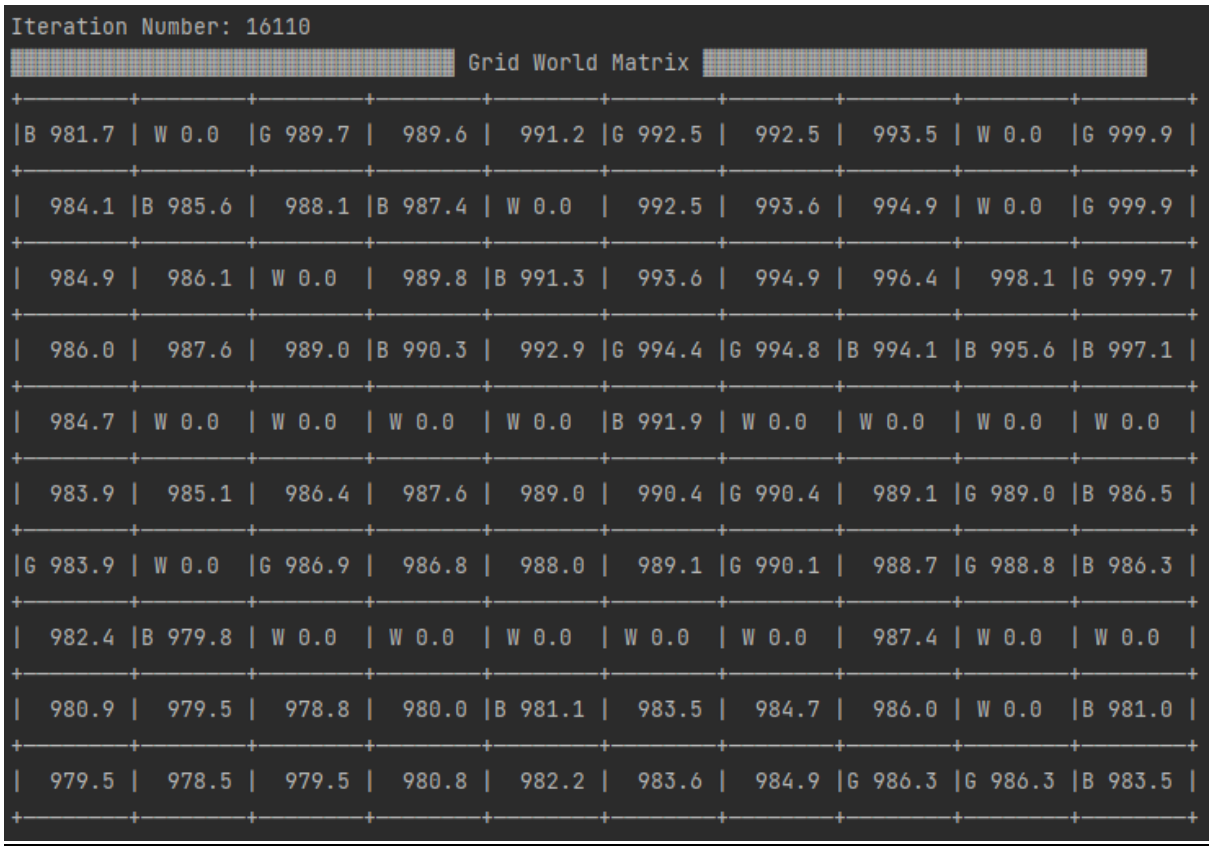
B 83.25	W 0.0	G 90.98	90.08	91.54	G 92.75	92.67	93.62	W 0.0	G 99.90
85.56	B 86.95	89.32	B 87.90	W 0.0	92.67	93.76	94.98	W 0.0	G 99.90
85.81	86.86	W 0.0	90.10	B 91.50	93.71	94.97	96.38	98.10	G 99.70
86.74	88.14	89.47	B 90.65	93.13	G 94.56	G 94.90	B 94.19	B 95.62	B 97.03
85.61	W 0.0	W 0.0	W 0.0	W 0.0	B 92.13	W 0.0	W 0.0	W 0.0	W 0.0
85.47	86.57	87.71	88.27	89.49	90.79	G 90.88	89.69	G 89.82	B 87.45
G 85.65	W 0.0	G 88.94	87.84	88.62	89.69	G 90.73	89.67	G 89.80	B 87.44
84.30	B 82.36	W 0.0	W 0.0	W 0.0	W 0.0	W 0.0	90.66	W 0.0	W 0.0
83.50	84.43	85.68	86.70	B 87.55	89.72	90.79	91.86	W 0.0	B 88.21
84.17	85.36	86.61	87.87	89.18	90.56	91.86	G 93.20	G 93.29	B 90.58

Appendix E

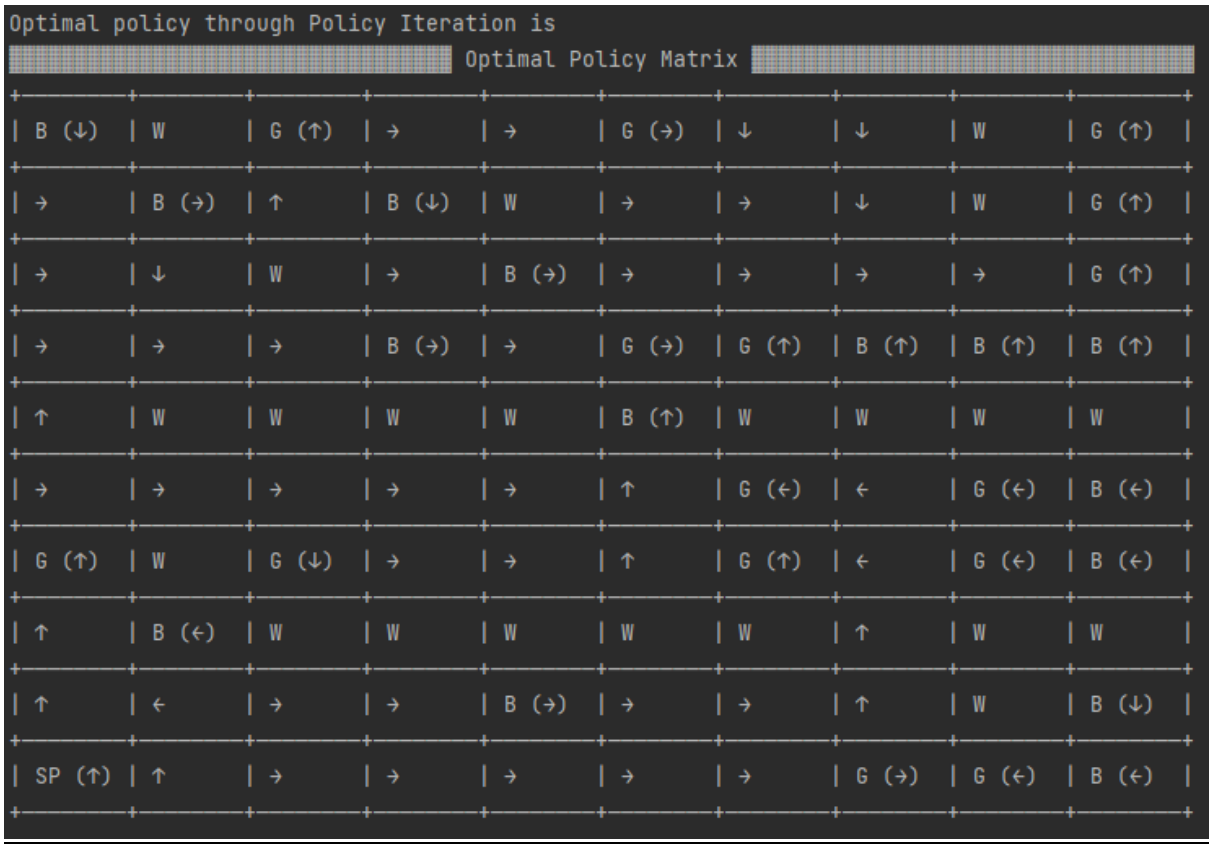
Optimal Policy for Value Iteration



Final Iteration Utility Values (16110) for Value Iteration



Optimal Policy for Policy Iteration



Final Policy and Utility Values (7)

Iteration 7

Optimal Policy Matrix

B (↓)	W	G (↑)	→	→	G (→)	↓	↓	W	G (↑)
→	B (→)	↑	B (↓)	W	→	→	↓	W	G (↑)
→	↓	W	→	B (→)	→	→	→	→	G (↑)
→	→	→	B (→)	→	G (→)	G (↑)	B (↑)	B (↑)	B (↑)
↑	W	W	W	W	B (↑)	W	W	W	W
→	→	→	→	→	↑	G (←)	←	G (←)	B (←)
G (↑)	W	G (↓)	→	→	↑	G (↑)	←	G (←)	B (←)
↑	B (←)	W	W	W	W	W	↑	W	W
↑	←	→	→	B (→)	→	→	↑	W	B (↓)
SP (↑)	↑	→	→	→	→	→	G (→)	G (←)	B (←)

Grid World Matrix

B 981.6	W 0.0	G 989.6	989.5	991.1	G 992.4	992.4	993.4	W 0.0	G 999.9
984.0	B 985.5	988.0	B 987.3	W 0.0	992.4	993.5	994.8	W 0.0	G 999.9
984.8	986.0	W 0.0	989.7	B 991.2	993.5	994.8	996.3	998.0	G 999.6
985.9	987.5	988.9	B 990.2	992.8	G 994.3	G 994.7	B 994.0	B 995.5	B 997.0
984.6	W 0.0	W 0.0	W 0.0	W 0.0	B 991.8	W 0.0	W 0.0	W 0.0	W 0.0
983.8	985.0	986.3	987.5	988.9	990.3	G 990.3	989.0	G 988.9	B 986.4
G 983.8	W 0.0	G 986.8	986.7	987.9	989.0	G 990.0	988.6	G 988.7	B 986.2
982.3	B 979.7	W 0.0	W 0.0	W 0.0	W 0.0	W 0.0	987.3	W 0.0	W 0.0
980.8	979.4	978.7	979.9	B 981.0	983.4	984.6	985.9	W 0.0	B 980.9
979.4	978.4	979.4	980.7	982.1	983.5	984.8	G 986.2	G 986.2	B 983.4