NANYANG
TECHNOLOGICAL
UNIVERSITY
SINGAPORE

CE/CZ4046 – Intelligent Agents

Assignment 1: Agent Decision Making

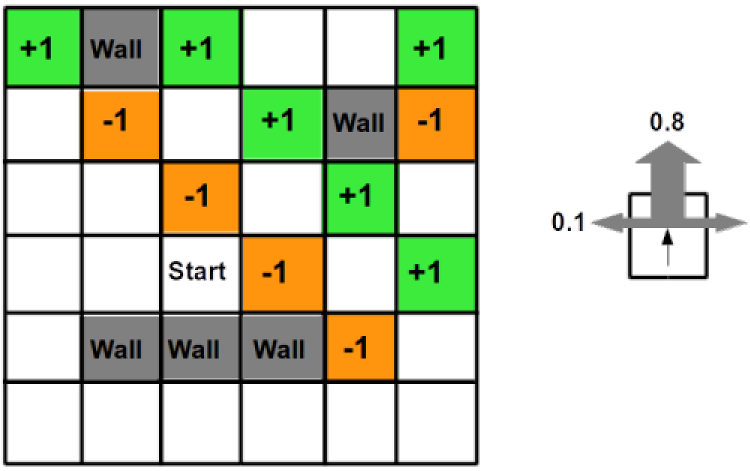
Name - Garg Astha

Matriculation No. - U1923971H

Table of Contents

1. Problem Statement

We have been given a maze environment. The transition model for which is as follows: the intended outcome occurs with probability 0.8, and with probability 0.1 the agent moves at either right angle to the intended direction. If the move would make the agent walk into a wall, the agent stays in the same place as before. The rewards for the white squares are -0.04, for the green squares are +1, and for the brown squares are -1. Note that there are no terminal states; the agent's state sequence is infinite.



+1	Wall	+1			+1
	-1		+1	Wall	-1
		-1		+1	
		Start	-1		+1
	Wall	Wall	Wall	-1	

Figure 1. Question (from the Assignment document)

With the given transition model and the reward function, the optimal policy, and the utilities of all the (non-wall) states using both value iteration and policy iteration must be calculated. In addition to this, the optimal policy, and the utilities of all the states are to be displayed, and the utility estimates as a function of the number of iterations are to be plotted. The discount factor of 0.990 is to be used for the purpose of this question.

2. Organization of the code

The code has been written in java language. The figure below displays the organization of the code files in different folders.

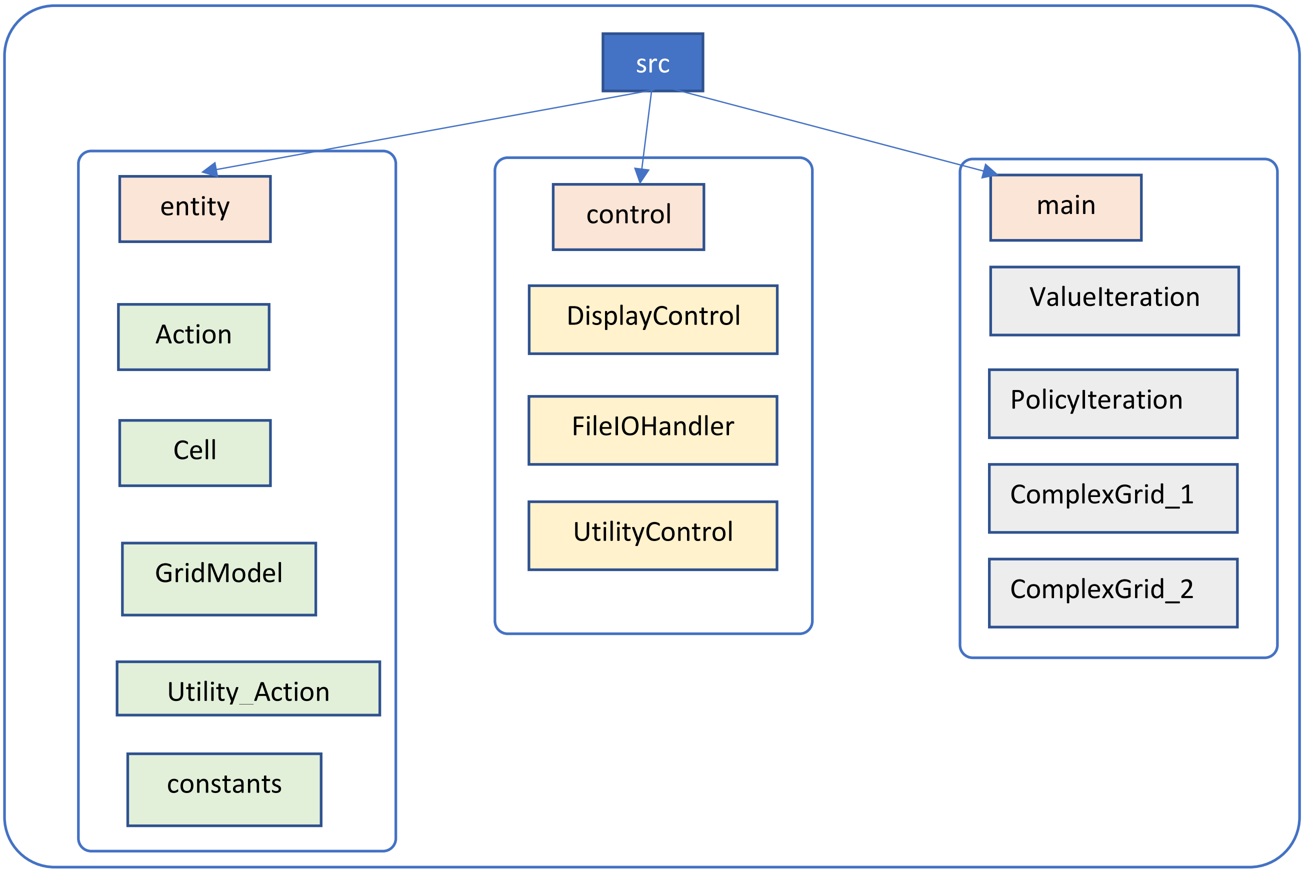


Figure 2. Code organization

Entity package:

1. Action – This class defines the four actions and enumerates them with symbolic representations.
2. Cell – This class will be used to create the grid later on. It has properties for reward for the cell and if the cell is a wall state.
3. GridModel – This creates a 2-D array of type cell and initializes the cells with white, brown and green rewards.
4. Utility_Action – This class helps in recording the utility and the corresponding action that the agent should take in a specific state.
5. Constants – This file defines the constants that are used in different files for various purposes. This includes the configuration for the value and policy iterations.

Control package:

1. DisplayControl – This class contains functions to display the grid, optimal policy and utilities of all states in a readable format.
2. FileIOHandler – This class contains functions to create csv with the utility values and save the optimal policy and utility values to a txt file.
3. UtilityControl – This class defines the different functions that are needed during the execution of the 2 iteration algorithms. It includes functions to estimate utilities and calculate and update utilities based on Bellman Equation etc.

Main package:

1. ValueIteration – This runs the value iteration algorithm by making use of the functions and values defined in other packages.
2. PolicyIteration - This runs the policy iteration algorithm by making use of the functions and values defined in other packages.
3. ComplexGrid_1 - This runs the value and policy iteration algorithm by making use of the functions and values defined in other packages for scale value of 3.
4. ComplexGrid_2 - This runs the value and policy iteration algorithm by making use of the functions and values defined in other packages for scale value of 6.

Other Files and Folders:

Plots.ipynb – This file in python is used to plot the graphs using the csv files obtained that store the utilities of the states.

Results – This folder contains subfolders to store all the csv, txt files and plot images. The subfolders "c_01", "c_1", "c_20" contain the results of value iteration algorithm for c values of 0.1, 1 and 20 respectively. The subfolders "k_10", "k_30" and "k_50" contain the results for policy iteration algorithm for k values of 10, 30 and 50 respectively.

The subfolder "complex grid results" contains the txt and csv files for the bonus question as well as the graphs.

The subfolder "images" contains the screenshots of plots for utility against iterations as obtained from the Plots.ipynb notebook.

3. Assumption

For the purpose of this assignment, it has been assumed that the agent can intend to move towards a wall or the boundary of grid. In these cases, it will try to move towards the wall (or boundary) with probability of 0.8 and to the left or right with probabilities of 0.1 each.

4. Setting up the Grid

The grid is set up as given in the question. To achieve this, an entity Cell has been defined which has a reward and a property to identify if it is a wall. The rewards are assigned as given in the question, with white squares (default value), having a value of -0.04, green squares a value of +1.0 and brown squares with -1.0. This initialization is common for both the value and policy iteration methods.

The GridModel is defined as a 2-D array of the type of Cell. They are first initialized with white squares by default. The cells with brown and green rewards are updated and then the wall cells are also updated.

This set up of the grid is used for executing value iteration, policy iteration of the given grid as well as running the algorithms on complex grid models.

5. Value Iteration

"The utility of a state is the immediate reward for that state plus the expected discounted utility of the next state, assuming that the agent chooses the optimal action.". That is, the utility of a state is given by the Bellman Equation as follows [1]:

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

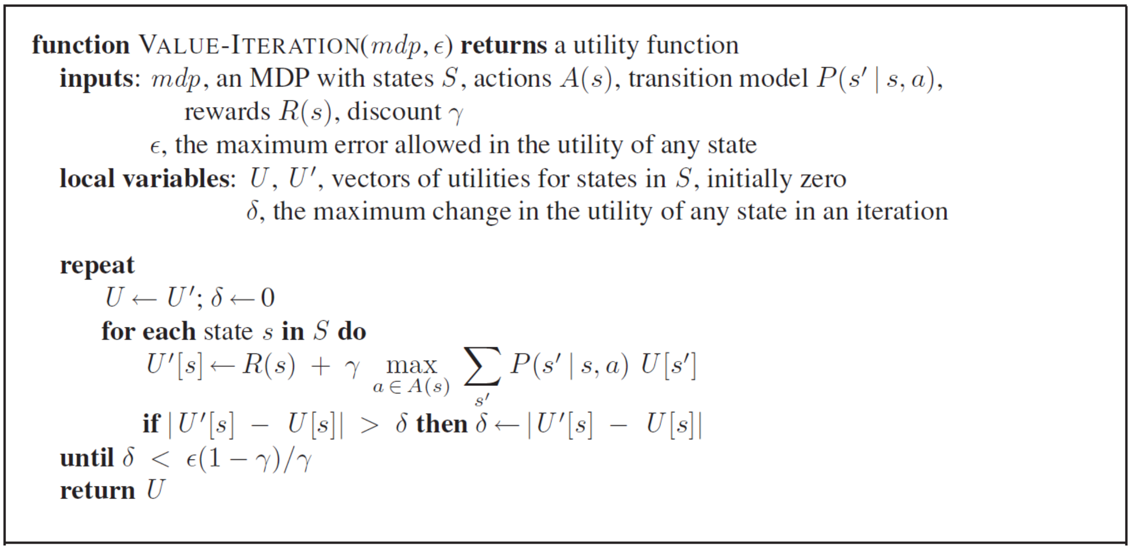
Equation 1. Bellman Equation [1]

The Bellman equation is the basis of the value iteration algorithm for solving Markov Decision Problems. There is one Bellman Equation for each state, thus, in the case of n possible states, then there are n Bellman equations.

The value iteration algorithm is implemented using an *iterative* approach, summarized in the figure 3. We start with arbitrary initial values for the utilities, calculate the right-hand side of the equation, and plug it into the left-hand side thereby updating the utility of each state from the utilities of its neighbours. We repeat this until we reach an equilibrium. Let $U_i(s)$ be the utility value for state ‘s’ at the ith iteration. The iteration step is called Bellman Update, the equation for which is [1]:

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

Equation 2. Bellman Update Equation [1]



```
function VALUE-ITERATION($mdp, \epsilon$) 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 $\gamma$
          $\epsilon$, the maximum error allowed in the utility of any state
  local variables: $U$, $U'$, vectors of utilities for states in $S$, initially zero
                   $\delta$, 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. Value Iteration Algorithm [1]

a. Set up

Some constants that will be used for the implementation are as mentioned below:

C = [20.00, 1.000, 0.100]

R max = 1.000

Epsilon = [20.00, 1.000, 0.100]

Discount = 0.990

Convergence threshold = [0.202, 0.010, 0.0010]

A difference in these constant values can produce different result for utilities and the policy, which will be explained and investigated in more detail later.

b. Implementing Value Iteration

The code implementation of value iteration is based on the algorithm as in figure 3. To start off the code implementation, firstly, a 2-D array for storing the utilities is created. The array is initialized with value of 0.0 for the utility and the action is set as null for all the cells in the grid.

To keep a track of the change in utility value, a variable delta is initialized with the least value possible (Double.MIN_VALUE).

Next, we loop over every cell in the grid and find the best action to take when the agent is in that cell such that the utility of the next state will be maximized. This is done for all the non-wall states. This is achieved by finding the utility for all four actions (up, down, left, and right) using the Bellman's equation. With the action given, utility is calculated and then the action with the maximum utility is selected. For every cell, we find the difference in the previous utility vs the updated utility. If this difference is greater than the delta initialized earlier then value of delta is updated to the new difference. The variable delta is updated to record the maximum difference in the utility for each iteration. The count for iterations is then increased by 1, when the whole grid has been processed once. The utility and action found in the iteration are recorded in an array to be accessed later on.

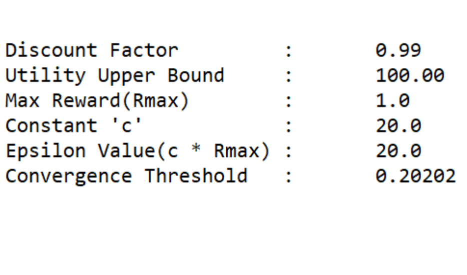
The above step of looping over the grid is repeated until delta is less than or equal to the convergence threshold. Since there is no time or iteration limit as well as terminating state, when delta is less or equal to the convergence threshold established earlier, the iterations are stopped. This also means that the change in utility for any cell in the grid will not be more than the allowed value (convergence threshold). The iterations over the grid are continued till this condition is not satisfied.

In the end, the array of the utilities is saved in a csv form to be plotted. ()

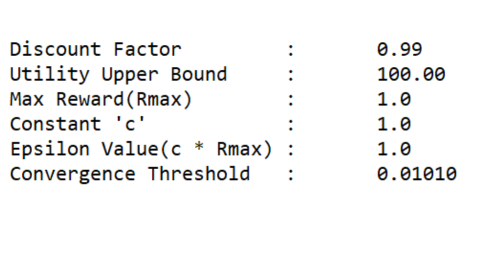
c. Results

i. Plot of optimal policy

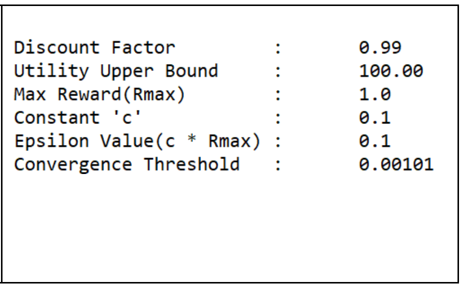
Table 1. Configuration for value iteration and plot of optimal policy



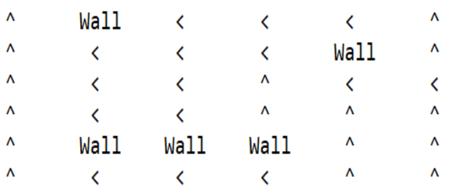
```
Discount Factor          :       0.99
Utility Upper Bound      :       100.00
Max Reward(Rmax)         :       1.0
Constant 'c'             :       20.0
Epsilon Value(c * Rmax)  :       20.0
Convergence Threshold    :       0.20202
```



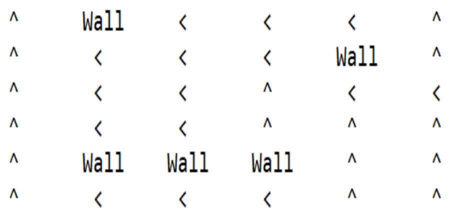
```
Discount Factor          :       0.99
Utility Upper Bound      :       100.00
Max Reward(Rmax)         :       1.0
Constant 'c'             :       1.0
Epsilon Value(c * Rmax)  :       1.0
Convergence Threshold    :       0.01010
```



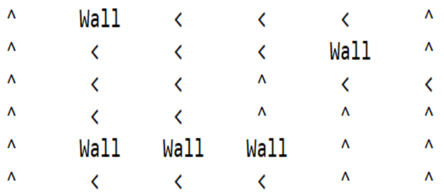
```
Discount Factor          :       0.99
Utility Upper Bound      :       100.00
Max Reward(Rmax)         :       1.0
Constant 'c'             :       0.1
Epsilon Value(c * Rmax)  :       0.1
Convergence Threshold    :       0.00101
```



```
^   Wall   <      <      <      ^
^   <      <      <      Wall   ^
^   <      <      ^      <      <
^   <      <      ^      ^      ^
^   Wall   Wall   Wall   ^      ^
^   <      <      <      ^      ^
```



```
^   Wall   <      <      <      ^
^   <      <      <      Wall   ^
^   <      <      ^      <      <
^   <      <      ^      ^      ^
^   Wall   Wall   Wall   ^      ^
^   <      <      <      ^      ^
```

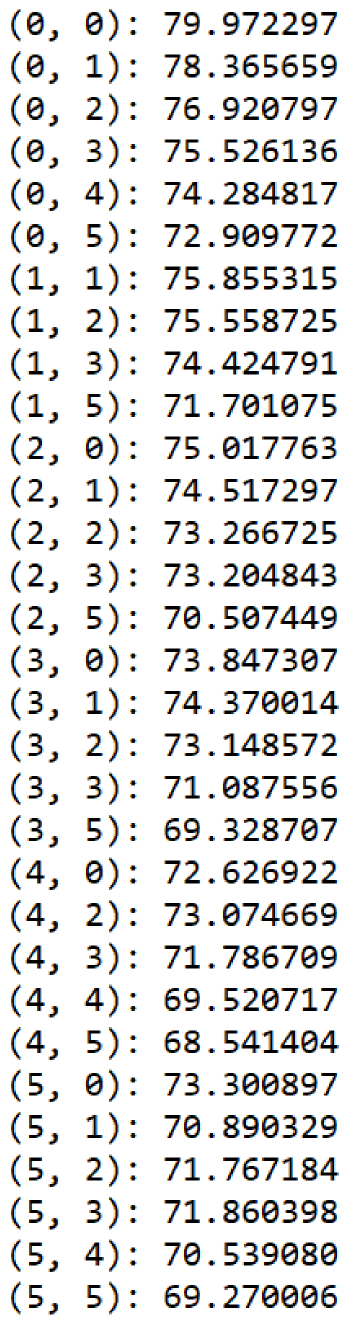


```
^   Wall   <      <      <      ^
^   <      <      <      Wall   ^
^   <      <      ^      <      <
^   <      <      ^      ^      ^
^   Wall   Wall   Wall   ^      ^
^   <      <      <      ^      ^
```

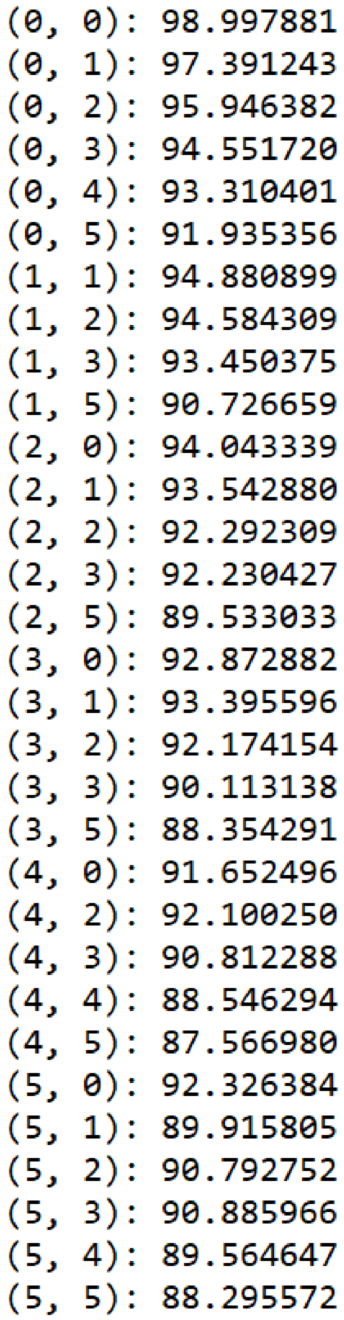
ii. Utilities of all states

Table 2. Utilities of all states

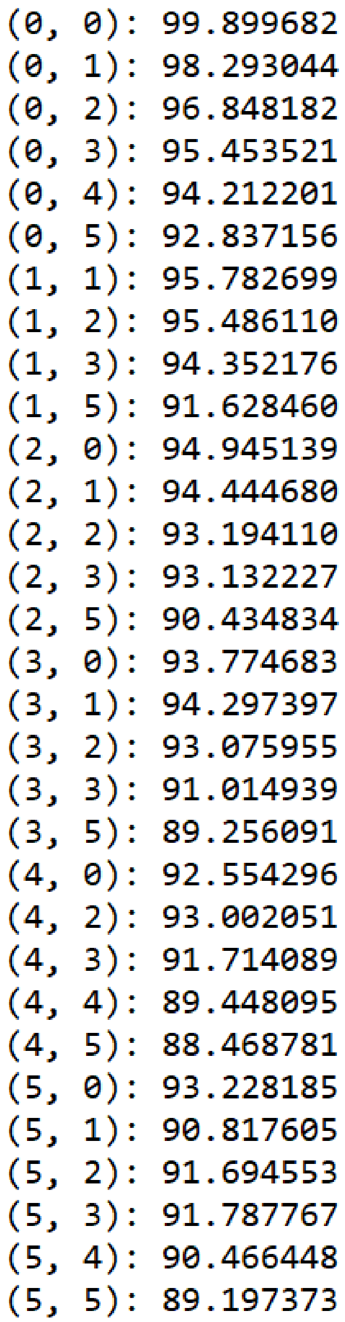
c = 20.0 Iterations: 161	c = 1.0 Iterations: 459	c = 0.1 Iterations: 688



```
(0, 0): 79.972297
(0, 1): 78.365659
(0, 2): 76.920797
(0, 3): 75.526136
(0, 4): 74.284817
(0, 5): 72.909772
(1, 1): 75.855315
(1, 2): 75.558725
(1, 3): 74.424791
(1, 5): 71.701075
(2, 0): 75.017763
(2, 1): 74.517297
(2, 2): 73.266725
(2, 3): 73.204843
(2, 5): 70.507449
(3, 0): 73.847307
(3, 1): 74.370014
(3, 2): 73.148572
(3, 3): 71.087556
(3, 5): 69.328707
(4, 0): 72.626922
(4, 2): 73.074669
(4, 3): 71.786709
(4, 4): 69.520717
(4, 5): 68.541404
(5, 0): 73.300897
(5, 1): 70.890329
(5, 2): 71.767184
(5, 3): 71.860398
(5, 4): 70.539080
(5, 5): 69.270006
```



```
(0, 0): 98.997881
(0, 1): 97.391243
(0, 2): 95.946382
(0, 3): 94.551720
(0, 4): 93.310401
(0, 5): 91.935356
(1, 1): 94.880899
(1, 2): 94.584309
(1, 3): 93.450375
(1, 5): 90.726659
(2, 0): 94.043339
(2, 1): 93.542880
(2, 2): 92.292309
(2, 3): 92.230427
(2, 5): 89.533033
(3, 0): 92.872882
(3, 1): 93.395596
(3, 2): 92.174154
(3, 3): 90.113138
(3, 5): 88.354291
(4, 0): 91.652496
(4, 2): 92.100250
(4, 3): 90.812288
(4, 4): 88.546294
(4, 5): 87.566980
(5, 0): 92.326384
(5, 1): 89.915805
(5, 2): 90.792752
(5, 3): 90.885966
(5, 4): 89.564647
(5, 5): 88.295572
```



```
(0, 0): 99.899682
(0, 1): 98.293044
(0, 2): 96.848182
(0, 3): 95.453521
(0, 4): 94.212201
(0, 5): 92.837156
(1, 1): 95.782699
(1, 2): 95.486110
(1, 3): 94.352176
(1, 5): 91.628460
(2, 0): 94.945139
(2, 1): 94.444680
(2, 2): 93.194110
(2, 3): 93.132227
(2, 5): 90.434834
(3, 0): 93.774683
(3, 1): 94.297397
(3, 2): 93.075955
(3, 3): 91.014939
(3, 5): 89.256091
(4, 0): 92.554296
(4, 2): 93.002051
(4, 3): 91.714089
(4, 4): 89.448095
(4, 5): 88.468781
(5, 0): 93.228185
(5, 1): 90.817605
(5, 2): 91.694553
(5, 3): 91.787767
(5, 4): 90.466448
(5, 5): 89.197373
```

The results are stored as follows:

- Configuration, utility values and plot of optimal policy
  - C = 0.1, CZ4046-Assignment1\GridWorld\results\c_01\config_results_values.txt
    CZ4046-Assignment1\GridWorld\results\c_01\value_iteration_utilities.csv
  - C = 1.0, CZ4046-Assignment1\GridWorld\results\c_1\config_results_values.txt

CZ4046-Assignment1\GridWorld\results\c_1\value_iteration_utilities.csv

  - C = 20.0, CZ4046-Assignment1\GridWorld\results\c_20\config_results_values.txt
    CZ4046-Assignment1\GridWorld\results\c_20\value_iteration_utilities.csv

- Plot of utilities of the states against the number of iterations
  - C = 0.1, CZ4046-Assignment1\GridWorld\results\images\value_c_01.png
  - C = 1.0, CZ4046-Assignment1\GridWorld\results\images\value_c_1.png
  - C = 20.0, CZ4046-Assignment1\GridWorld\results\images\value_c_20.png

iii. Findings

Executing value iteration for different c values returns the same optimal policy. Although the utility values are different because of different number of iterations. Higher the value of c, higher is the convergence threshold which means that convergence is achieved with lesser number of iterations.
It can also be seen from the plots below (figures 4 - 6) that the change in utility is insignificant after about 250 iterations. It can be inferred that the optimal policy was learned by that number of iterations however, the iterations continued as the convergence threshold was not yet achieved.

d. Plot of utility estimates as function of number of iterations

The code for the plots is in the file Plots.ipynb.

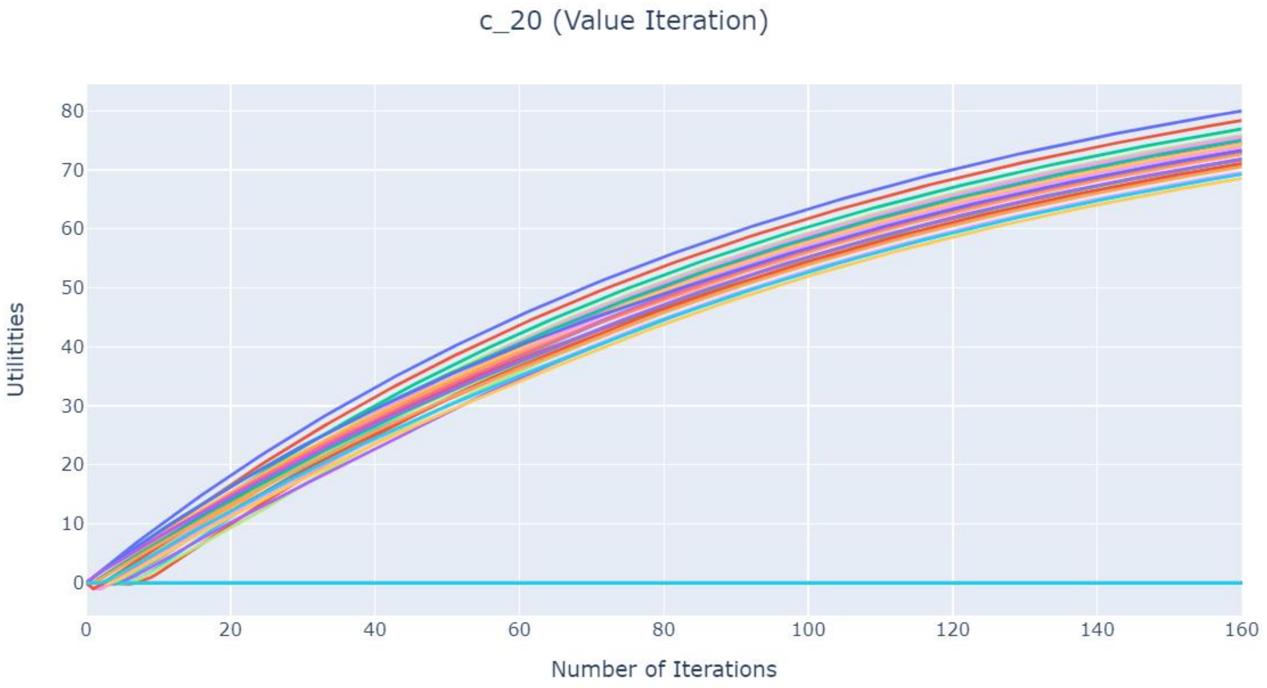


Figure 4. Value iteration for c = 20

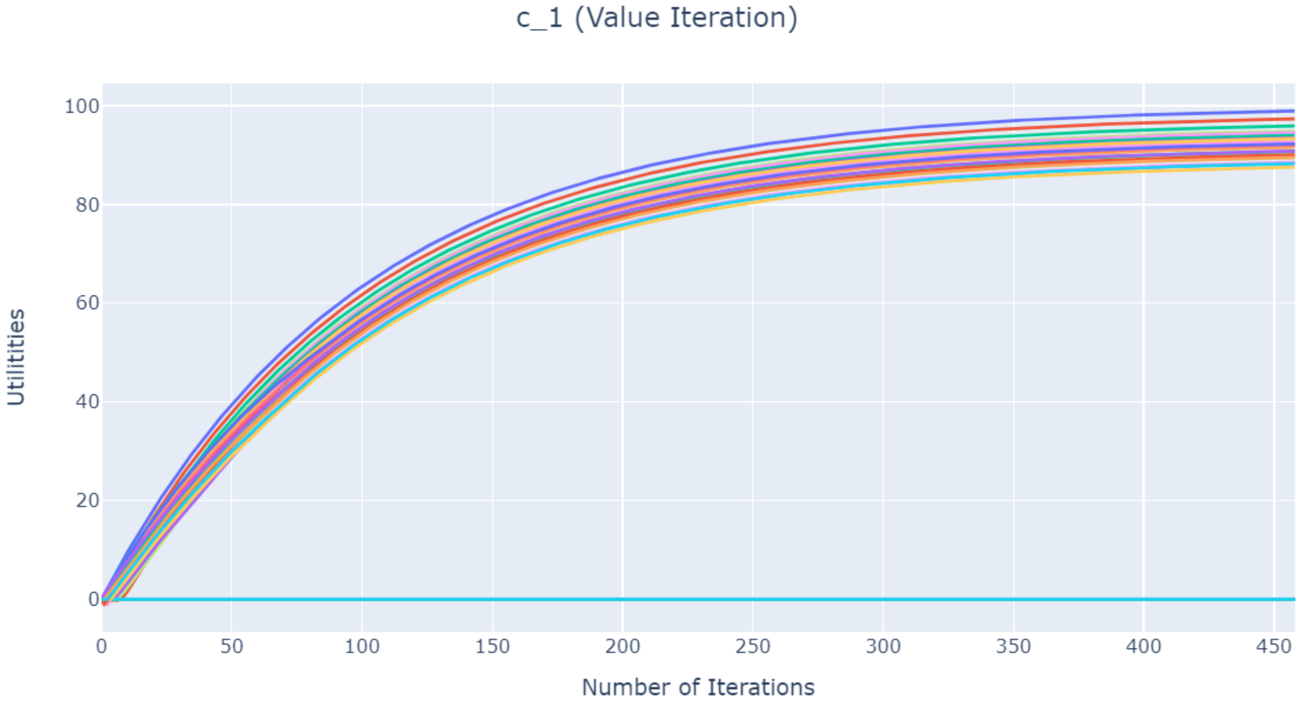


Figure 5. Value Iteration for c = 1

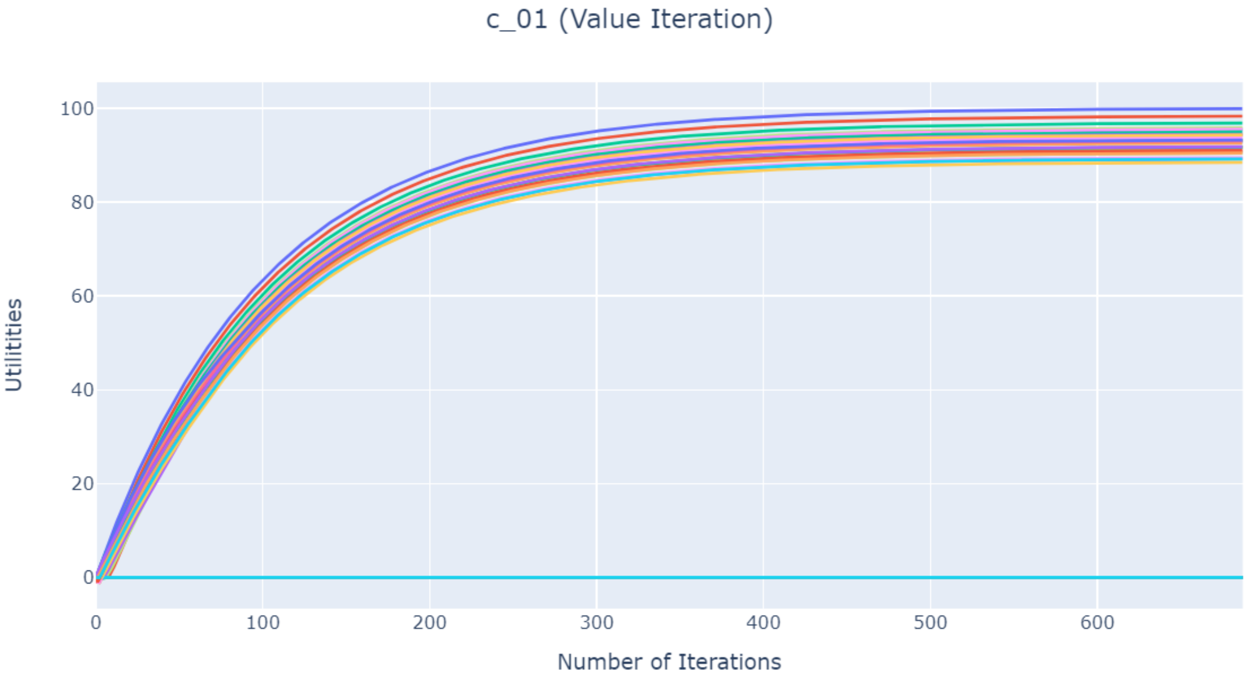


Figure 6. Value Iteration for c = 0.1

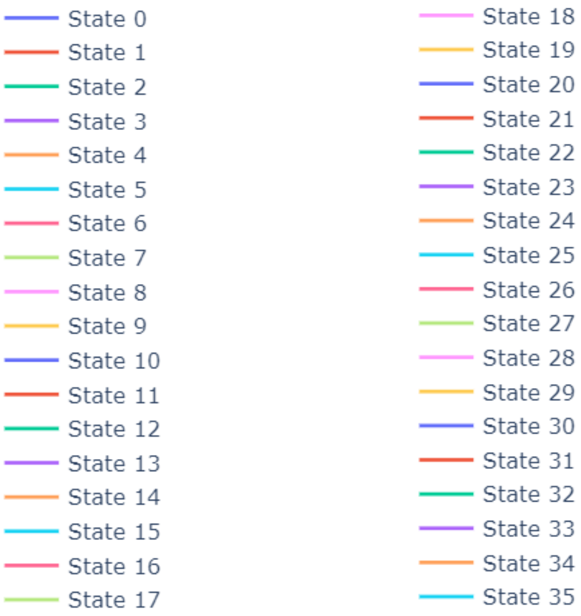


Figure 7. Legend for the plots

6. Policy Iteration

We saw in the result of the value iteration that it is possible to get an optimal policy even when the utility function estimate is inaccurate. If one action is clearly better than all others, then the exact magnitude of the utilities on the states involved need not be precise. The **policy iteration** algorithm alternates the following two steps, beginning from some initial policy π_0 [1]:

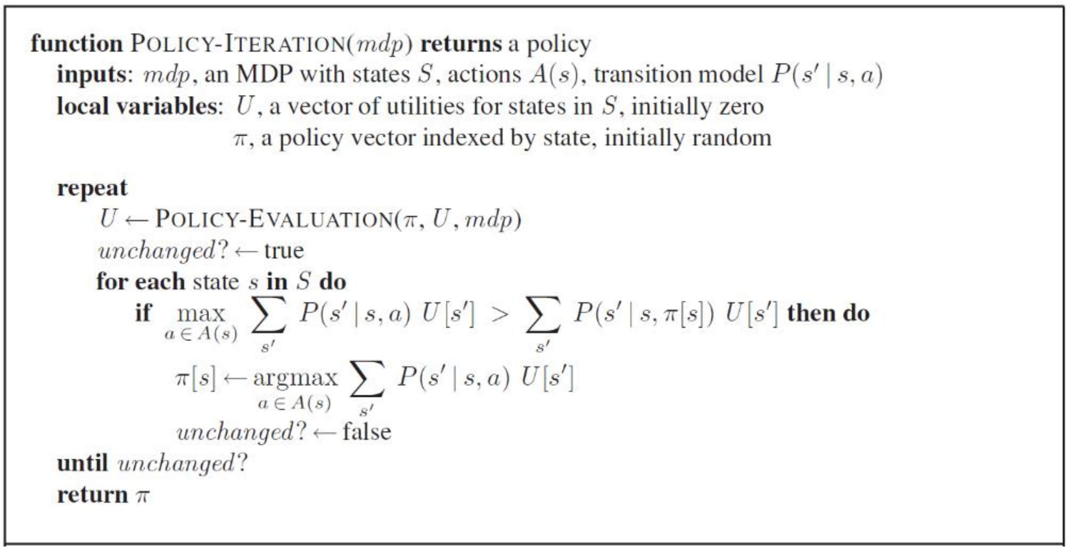
- **Policy evaluation**: given a policy π_i , calculate Ui =U π_i , the utility of each state if π_i were to be executed.
- **Policy improvement**: Calculate a new MEU policy π_{i+1}, using one-step look-ahead based on U_i.

The algorithm terminates when the policy improvement step yields no change in the utilities. This means that we have a simplified version of the Bellman equation relating the utility of s (under π_i) to the utilities of its neighbours [1]:

$$U_i(s) = R(s) + \gamma \sum_{s'} P(s' \mid s, \pi_i(s)) U_i(s') \ .$$

Equation 3. Simplified Bellman Equation

It is not necessary to do *exact* policy evaluation. Instead, we can perform some number of simplified value iteration steps (simplified because the policy is fixed) to give a reasonably good approximation of the utilities. The simplified Bellman update for this process



```
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
                   $\pi$, a policy vector indexed by state, initially random

  repeat
     $U \leftarrow$ POLICY-EVALUATION($\pi, U, mdp$)
     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 $\pi$
```

Figure 8. Policy Iteration Algorithm

is repeated k times to produce the next utility estimate. The resulting algorithm is called **modified policy iteration**. It is often much more efficient than standard policy iteration or value iteration [1].

a. Set up

The k values used for the policy iteration are as follows:
k = [10, 30, 50]

b. Implementing policy iteration

The code for implementation of policy iteration algorithm is the modified policy iteration method as described earlier. Firstly, a utility and action 2-D array is declared with the values set to random actions for the initial policy. A Boolean variable is initialized to keep track of when the policy stops changing. It is set to false before iterating over the cells of the grid. Another array is declared to save the utilities of the state after each complete iteration which will be used to plot the graph for later.

On the basis of the current utility array, new utility values are estimated by running the simplified Bellman Equation for all the cells of the grid k number of times. This is in accordance with the algorithm mentioned above to get a good approximation of the utilities. For each of the k iterations, a utility array is created to hold the current policy and the utility of the states based on this policy.

After finding the utility estimates using the simplified bellman update equations, we find the best action that will maximize the utility for the subsequent states. This is done by finding the utility of all the four actions as done in Value Iteration. Then we choose the action that will maximize the utility for the subsequent states in the grid.

Next, for all the non-wall cells in the grid, the utility of the action obtained from the current policy is compared with the utility of the best action found in the previous step. If the utility of the best action is greater than that given by the current policy, then the policy is updated, and the Boolean variable tracking change is updated.

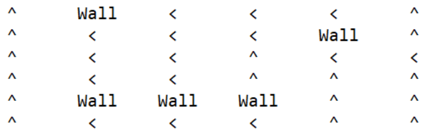
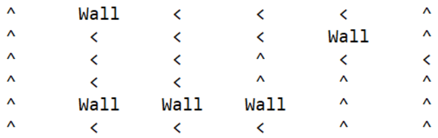
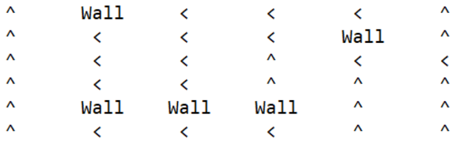
This process is repeated till the policy for all the cells is unchanged. This simply implies that the taking the actions defined by the current policy bring no change in the utility as compared to the best action.

In the end, the array of the utilities is saved in a csv form to be plotted.

c. Results

i. Plot of optimal policy

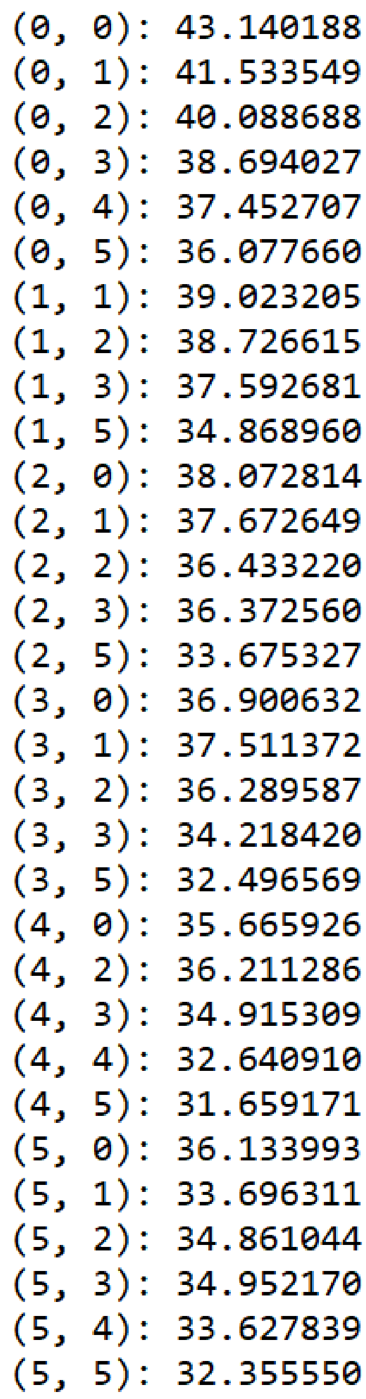
Table 3. Configuration of Policy Iteration and plot of optimal policy

<table>
<tr><th>K = 10</th><th>K = 30</th><th>K = 50</th></tr>
<tr>
<td>
<pre>
^   Wall   <   <   <   ^
^   <   <   <   Wall   ^
^   <   <   ^   <   <
^   <   <   ^   ^   ^
^   Wall Wall Wall   ^   ^
^   <   <   <   ^   ^
</pre></td>
<td>
<pre>
^   Wall   <   <   <   ^
^   <   <   <   Wall   ^
^   <   <   ^   <   <
^   <   <   ^   ^   ^
^   Wall Wall Wall   ^   ^
^   <   <   <   ^   ^
</pre></td>
<td>
<pre>
^   Wall   <   <   <   ^
^   <   <   <   Wall   ^
^   <   <   ^   <   <
^   <   <   ^   ^   ^
^   Wall Wall Wall   ^   ^
^   <   <   <   ^   ^
</pre></td>
</tr>
</table>

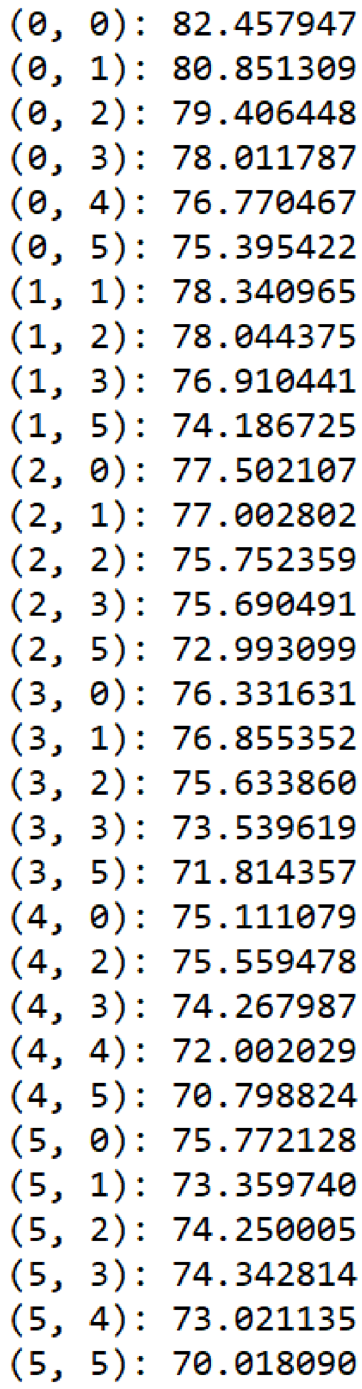
ii. Utilities of all states

Table 4. utilities of all states

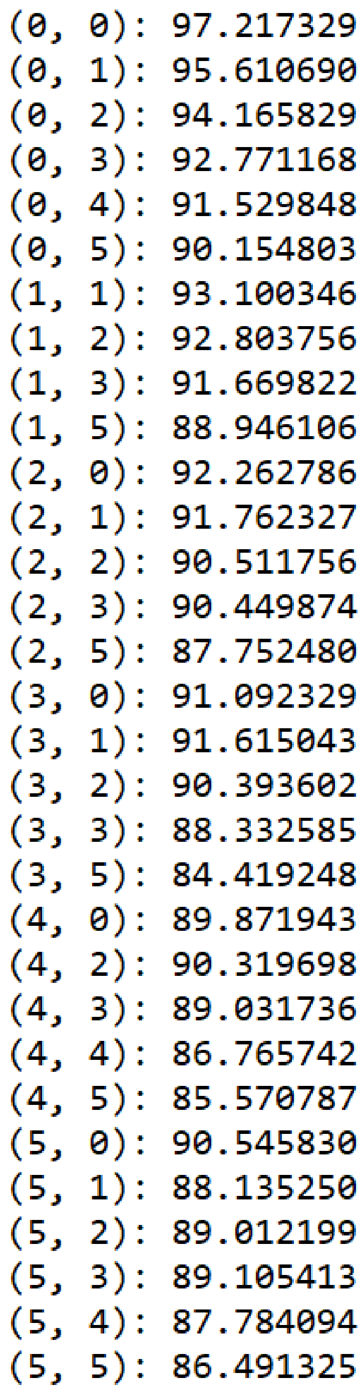
K = 10 Iterations: 7	K = 30 Iterations: 7	K = 50 Iterations: 9



```
(0, 0): 43.140188
(0, 1): 41.533549
(0, 2): 40.088688
(0, 3): 38.694027
(0, 4): 37.452707
(0, 5): 36.077660
(1, 1): 39.023205
(1, 2): 38.726615
(1, 3): 37.592681
(1, 5): 34.868960
(2, 0): 38.072814
(2, 1): 37.672649
(2, 2): 36.433220
(2, 3): 36.372560
(2, 5): 33.675327
(3, 0): 36.900632
(3, 1): 37.511372
(3, 2): 36.289587
(3, 3): 34.218420
(3, 5): 32.496569
(4, 0): 35.665926
(4, 2): 36.211286
(4, 3): 34.915309
(4, 4): 32.640910
(4, 5): 31.659171
(5, 0): 36.133993
(5, 1): 33.696311
(5, 2): 34.861044
(5, 3): 34.952170
(5, 4): 33.627839
(5, 5): 32.355550
```



```
(0, 0): 82.457947
(0, 1): 80.851309
(0, 2): 79.406448
(0, 3): 78.011787
(0, 4): 76.770467
(0, 5): 75.395422
(1, 1): 78.340965
(1, 2): 78.044375
(1, 3): 76.910441
(1, 5): 74.186725
(2, 0): 77.502107
(2, 1): 77.002802
(2, 2): 75.752359
(2, 3): 75.690491
(2, 5): 72.993099
(3, 0): 76.331631
(3, 1): 76.855352
(3, 2): 75.633860
(3, 3): 73.539619
(3, 5): 71.814357
(4, 0): 75.111079
(4, 2): 75.559478
(4, 3): 74.267987
(4, 4): 72.002029
(4, 5): 70.798824
(5, 0): 75.772128
(5, 1): 73.359740
(5, 2): 74.250005
(5, 3): 74.342814
(5, 4): 73.021135
(5, 5): 70.018090
```



```
(0, 0): 97.217329
(0, 1): 95.610690
(0, 2): 94.165829
(0, 3): 92.771168
(0, 4): 91.529848
(0, 5): 90.154803
(1, 1): 93.100346
(1, 2): 92.803756
(1, 3): 91.669822
(1, 5): 88.946106
(2, 0): 92.262786
(2, 1): 91.762327
(2, 2): 90.511756
(2, 3): 90.449874
(2, 5): 87.752480
(3, 0): 91.092329
(3, 1): 91.615043
(3, 2): 90.393602
(3, 3): 88.332585
(3, 5): 84.419248
(4, 0): 89.871943
(4, 2): 90.319698
(4, 3): 89.031736
(4, 4): 86.765742
(4, 5): 85.570787
(5, 0): 90.545830
(5, 1): 88.135250
(5, 2): 89.012199
(5, 3): 89.105413
(5, 4): 87.784094
(5, 5): 86.491325
```

The results are stored as follows:

- Configuration, utility values and plot of optimal policy
  - k = 10, CZ4046-Assignment1\GridWorld\results\k_10\config_results_policy.txt
    CZ4046-Assignment1\GridWorld\results\k_10\policy_iteration_utilities.csv

- k = 30, CZ4046-Assignment1\GridWorld\results\k_30\config_results_policy.txt
  CZ4046-Assignment1\GridWorld\results\k_30\policy_iteration_utilities.csv
- k = 50, CZ4046-Assignment1\GridWorld\results\k_50\config_results_policy.txt
  CZ4046-Assignment1\GridWorld\results\k_50\policy_iteration_utilities.csv

- Plot of utilities of the states against the number of iterations
  - K = 10, CZ4046-Assignment1\GridWorld\results\images\policy_k_10.png
  - K = 30, CZ4046-Assignment1\GridWorld\results\images\policy_k_30.png
  - K = 50, CZ4046-Assignment1\GridWorld\results\images\policy_k_50.png

iii. Findings

In this case as well, the different values of k give the same optimal policy. The number of iterations is much lower than that needed by value iteration. The number of iterations remains less than 10. The results for the utility can vary slightly as they are dependent on the initial random policy.

d. Plot of utility estimates as function of number of iterations

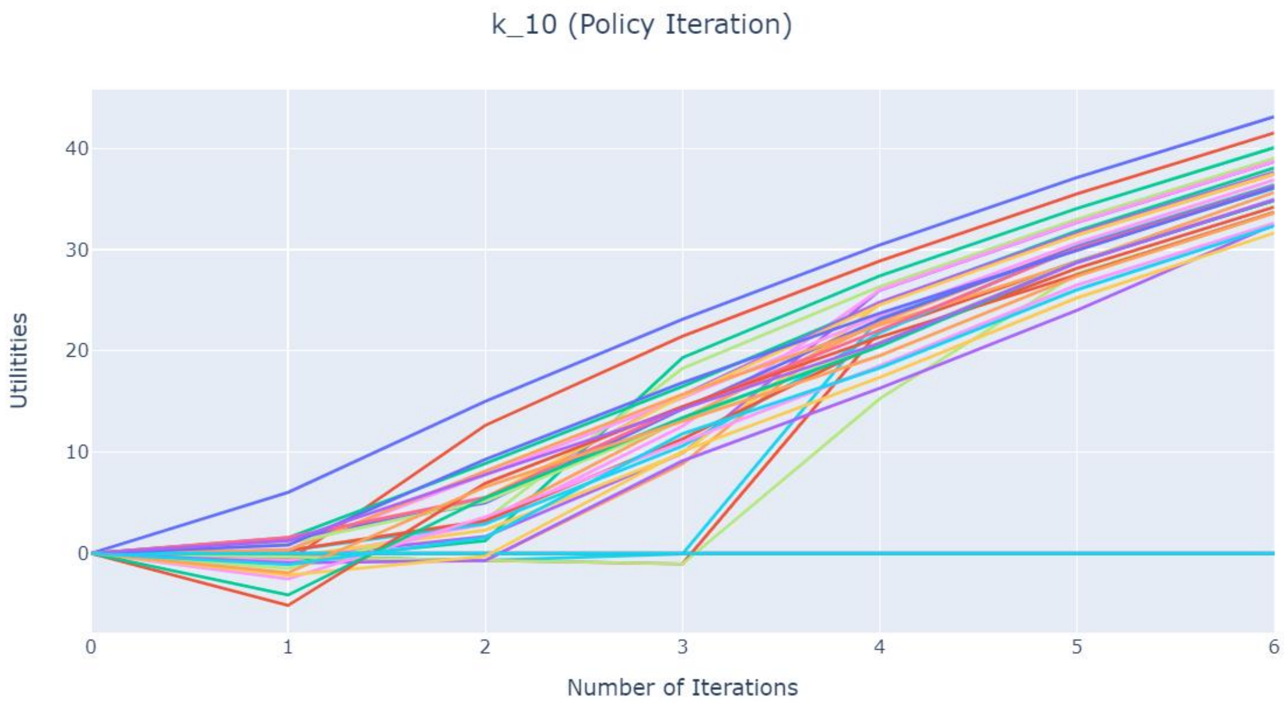


Figure 9. Policy Iteration for k = 10

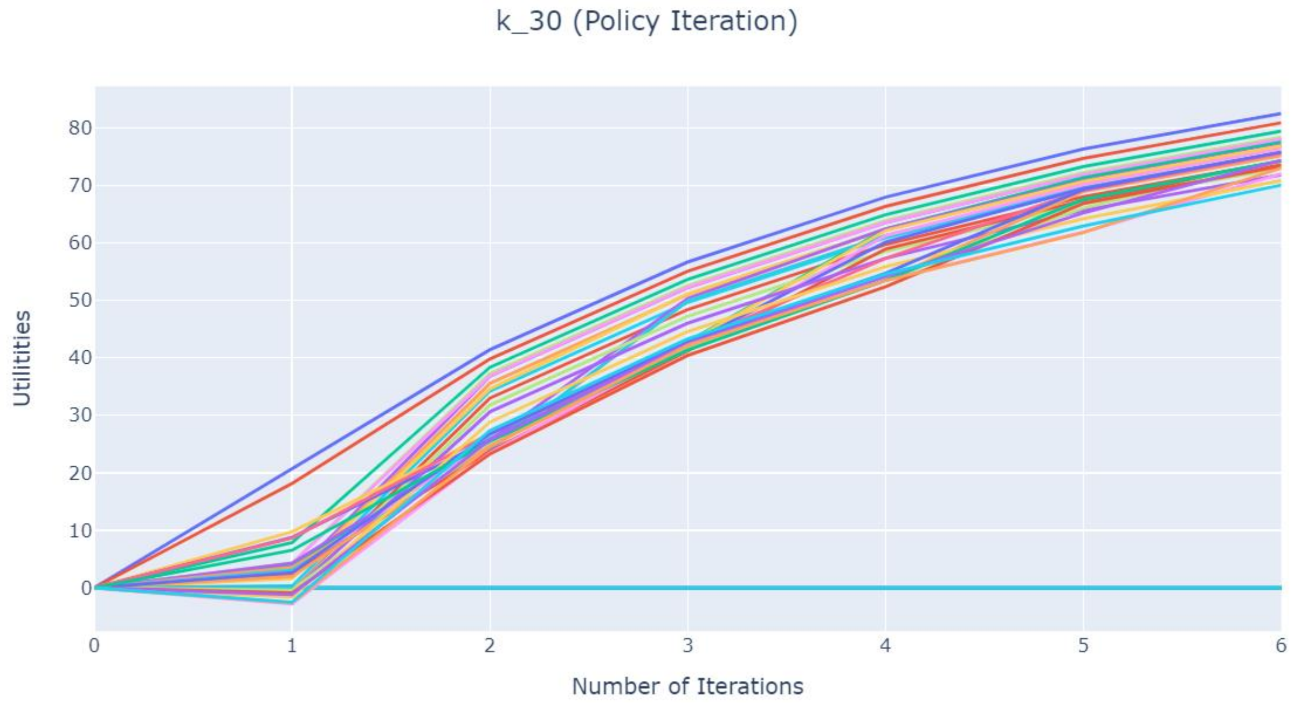


Figure 10. Policy Iteration for k = 30

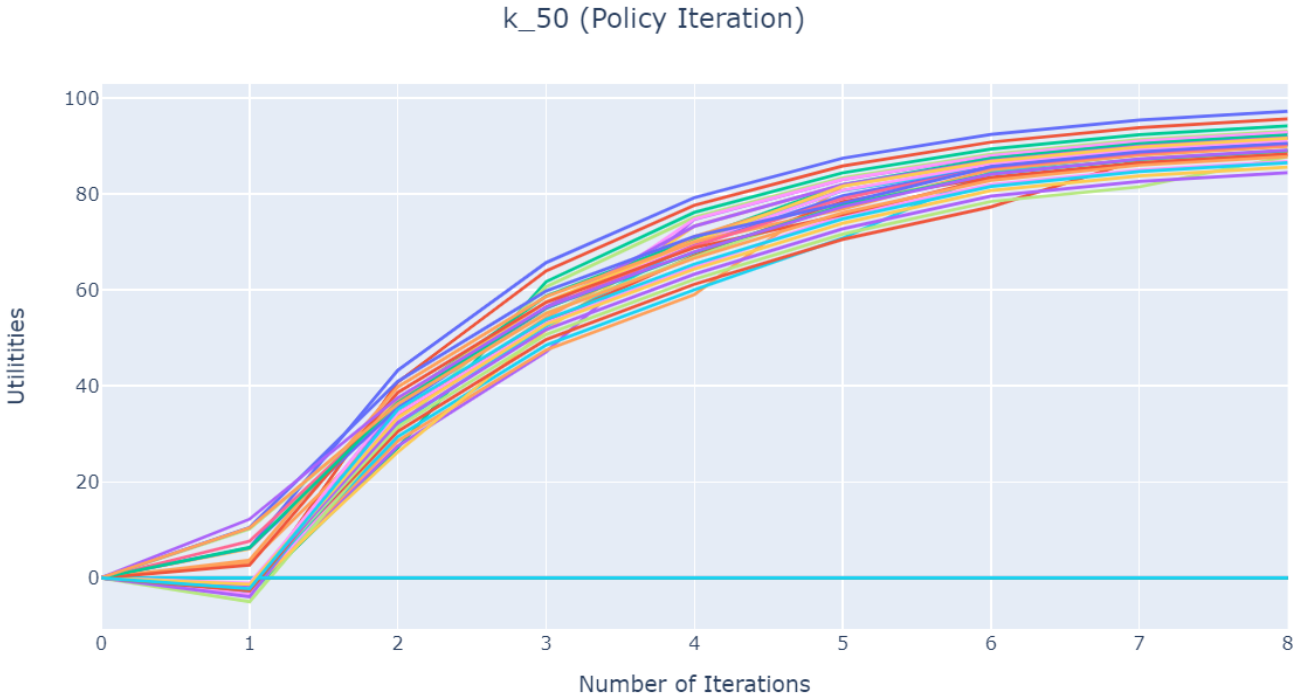


Figure 11. Policy Iteration for k = 50

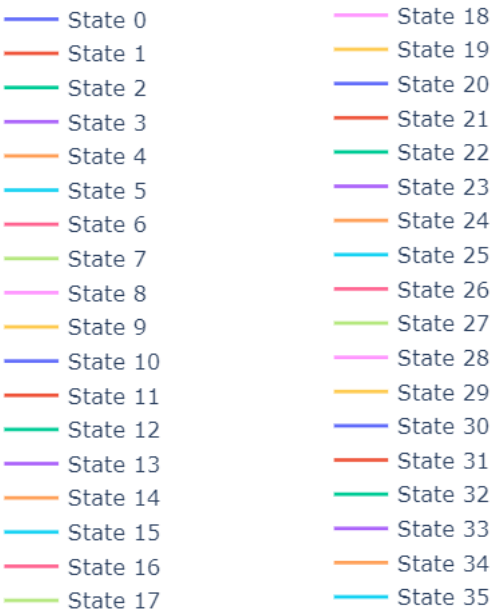
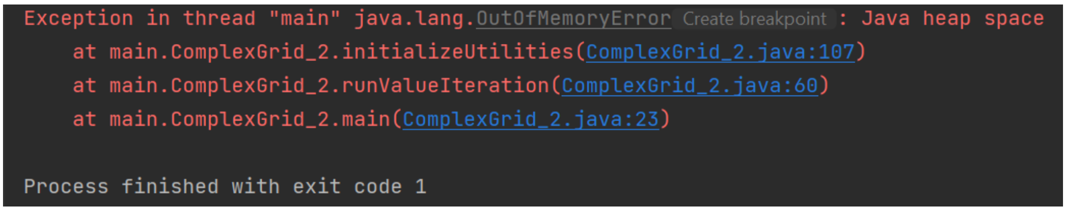


Figure 12. Legend for the plots

7. Complex Grid Environment

For constructing a more complex grid environment, the given grid is scaled up by factors of 3, 6 and 1000. This means the total number of states become 18*18, 36*36, 6000*6000 respectively.

In case of scale = 1000 i.e., 36,000,000 states, the program was not able to execute as the heap memory run out. This means that another approach to implement the algorithm is required as the current approach is inadequate.

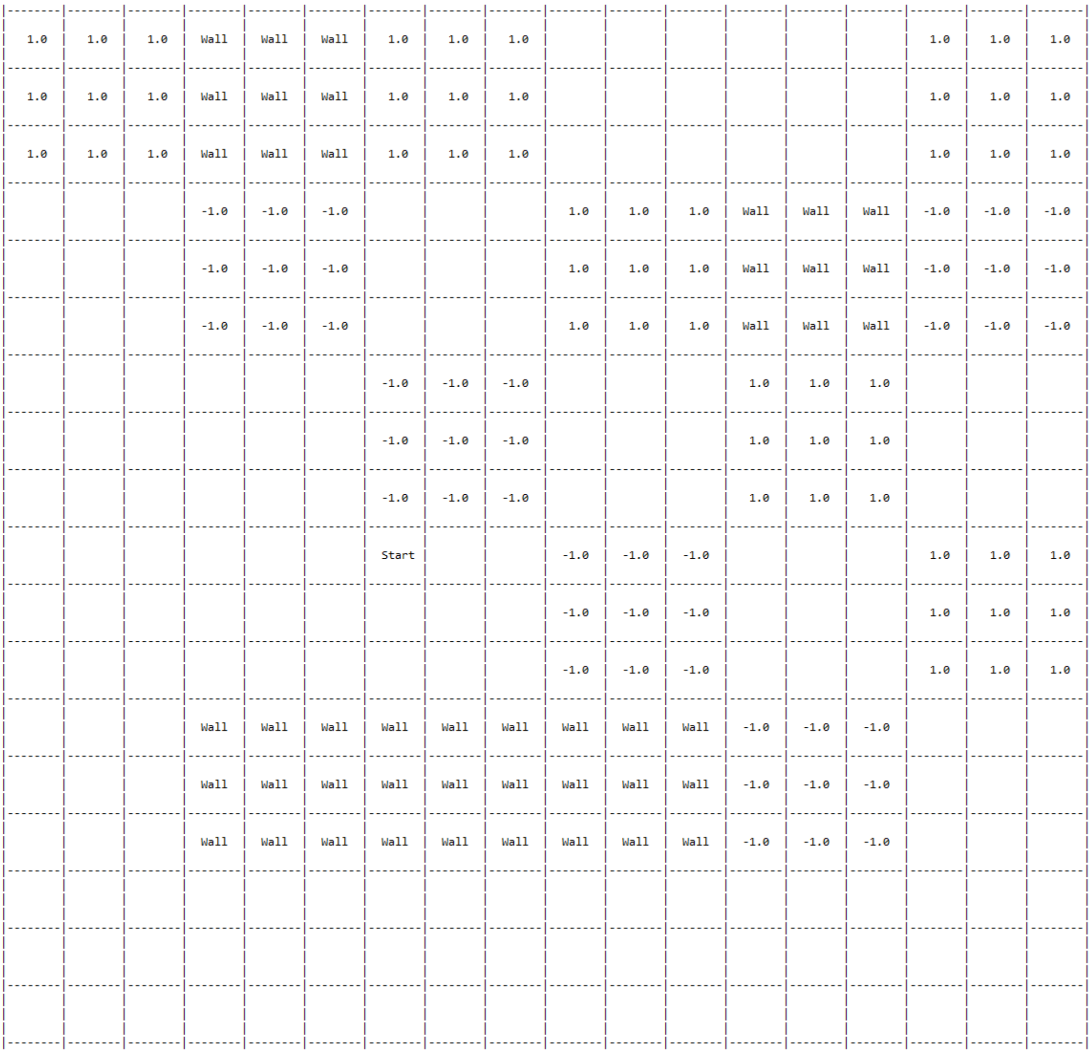


```
Exception in thread "main" java.lang.OutOfMemoryError Create breakpoint : Java heap space
	at main.ComplexGrid_2.initializeUtilities(ComplexGrid_2.java:107)
	at main.ComplexGrid_2.runValueIteration(ComplexGrid_2.java:60)
	at main.ComplexGrid_2.main(ComplexGrid_2.java:23)

Process finished with exit code 1
```

Figure 13. Exception in java

When scale = 3, the grid environment is initialized as follows:



```
|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
|  1.0   |  1.0   |  1.0   |  Wall  |  Wall  |  Wall  |  1.0   |  1.0   |  1.0   |        |        |        |        |        |        |  1.0   |  1.0   |  1.0   |
|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
|  1.0   |  1.0   |  1.0   |  Wall  |  Wall  |  Wall  |  1.0   |  1.0   |  1.0   |        |        |        |        |        |        |  1.0   |  1.0   |  1.0   |
|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
|  1.0   |  1.0   |  1.0   |  Wall  |  Wall  |  Wall  |  1.0   |  1.0   |  1.0   |        |        |        |        |        |        |  1.0   |  1.0   |  1.0   |
|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
|        |        |        |  -1.0  |  -1.0  |  -1.0  |        |        |        |  1.0   |  1.0   |  1.0   |  Wall  |  Wall  |  Wall  |  -1.0  |  -1.0  |  -1.0  |
|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
|        |        |        |  -1.0  |  -1.0  |  -1.0  |        |        |        |  1.0   |  1.0   |  1.0   |  Wall  |  Wall  |  Wall  |  -1.0  |  -1.0  |  -1.0  |
|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
|        |        |        |  -1.0  |  -1.0  |  -1.0  |        |        |        |  1.0   |  1.0   |  1.0   |  Wall  |  Wall  |  Wall  |  -1.0  |  -1.0  |  -1.0  |
|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
|        |        |        |        |        |        |  -1.0  |  -1.0  |  -1.0  |        |        |        |  1.0   |  1.0   |  1.0   |        |        |        |
|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
|        |        |        |        |        |        |  -1.0  |  -1.0  |  -1.0  |        |        |        |  1.0   |  1.0   |  1.0   |        |        |        |
|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
|        |        |        |        |        |        |  -1.0  |  -1.0  |  -1.0  |        |        |        |  1.0   |  1.0   |  1.0   |        |        |        |
|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
|        |        |        |        |        |        | Start  |        |        |  -1.0  |  -1.0  |  -1.0  |        |        |        |  1.0   |  1.0   |  1.0   |
|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
|        |        |        |        |        |        |        |        |        |  -1.0  |  -1.0  |  -1.0  |        |        |        |  1.0   |  1.0   |  1.0   |
|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
|        |        |        |        |        |        |        |        |        |  -1.0  |  -1.0  |  -1.0  |        |        |        |  1.0   |  1.0   |  1.0   |
|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
|        |        |        |  Wall  |  Wall  |  Wall  |  Wall  |  Wall  |  Wall  |  Wall  |  Wall  |  Wall  |  -1.0  |  -1.0  |  -1.0  |        |        |        |
|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
|        |        |        |  Wall  |  Wall  |  Wall  |  Wall  |  Wall  |  Wall  |  Wall  |  Wall  |  Wall  |  -1.0  |  -1.0  |  -1.0  |        |        |        |
|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
|        |        |        |  Wall  |  Wall  |  Wall  |  Wall  |  Wall  |  Wall  |  Wall  |  Wall  |  Wall  |  -1.0  |  -1.0  |  -1.0  |        |        |        |
|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
|        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |
|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
|        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |
|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
|        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |
|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
```

Figure 14. Complex grid model with scale = 3

In the above figure, the cells with +1.0 rewards are green squares, -1.0 are brown squares and white squares have -0.4 reward.

Policy iteration and value iteration is executed in the similar way as done earlier. The results for which are discussed as follows:

Table 5. Policy Iteration result for scale = 3

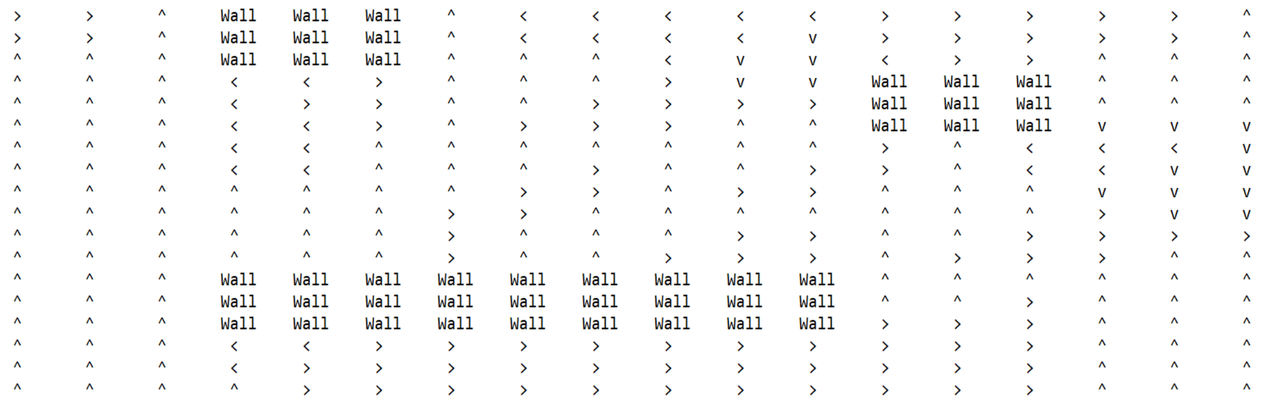
Policy Iteration

Scale = 3

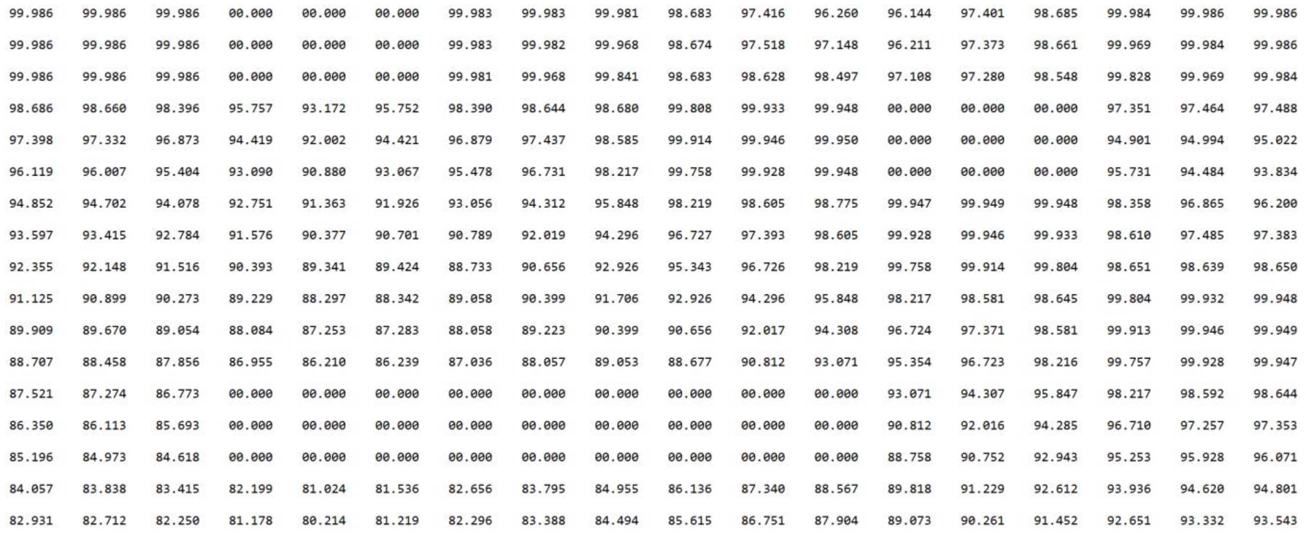
Discount Factor = 0.990

K = 50

Iterations = 19



```
 >     >     ^    Wall  Wall  Wall   ^     <     <     <     <     <     >     >     >     >     >     ^
 >     >     ^    Wall  Wall  Wall   ^     <     <     <     <     v     >     >     >     >     >     ^
 ^     ^     ^    Wall  Wall  Wall   ^     ^     ^     <     v     v     <     >     >     ^     ^     ^
 ^     ^     ^     <     <     >     ^     ^     ^     >     v     v    Wall  Wall  Wall   ^     ^     ^
 ^     ^     ^     <     >     >     ^     ^     >     >     >     >    Wall  Wall  Wall   ^     ^     ^
 ^     ^     ^     <     <     >     ^     >     >     >     ^     ^    Wall  Wall  Wall   v     v     v
 ^     ^     ^     <     <     ^     ^     ^     ^     ^     ^     ^     >     ^     <     <     <     v
 ^     ^     ^     <     <     ^     ^     ^     >     ^     ^     >     >     ^     <     <     v     v
 ^     ^     ^     ^     ^     ^     ^     >     >     ^     >     >     ^     ^     ^     v     v     v
 ^     ^     ^     ^     ^     ^     >     >     ^     ^     ^     ^     ^     ^     ^     >     v     v
 ^     ^     ^     ^     ^     ^     >     ^     ^     ^     >     >     ^     ^     >     >     >     >
 ^     ^     ^     ^     ^     ^     >     ^     ^     >     >     >     ^     >     >     >     ^     ^
 ^     ^     ^    Wall  Wall  Wall  Wall  Wall  Wall  Wall  Wall  Wall   ^     ^     ^     ^     ^     ^
 ^     ^     ^    Wall  Wall  Wall  Wall  Wall  Wall  Wall  Wall  Wall   ^     ^     >     ^     ^     ^
 ^     ^     ^    Wall  Wall  Wall  Wall  Wall  Wall  Wall  Wall  Wall   >     >     >     ^     ^     ^
 ^     ^     ^     <     <     >     >     >     >     >     >     >     >     >     >     ^     ^     ^
 ^     ^     ^     <     >     >     >     >     >     >     >     >     >     >     >     ^     ^     ^
 ^     ^     ^     ^     >     >     >     >     >     >     >     >     >     >     >     ^     ^     ^
```



```
99.986  99.986  99.986  00.000  00.000  00.000  99.983  99.983  99.981  98.683  97.416  96.260  96.144  97.401  98.685  99.984  99.986  99.986
99.986  99.986  99.986  00.000  00.000  00.000  99.983  99.982  99.968  98.674  97.518  97.148  96.211  97.373  98.661  99.969  99.984  99.986
99.986  99.986  99.986  00.000  00.000  00.000  99.981  99.968  99.841  98.683  98.628  98.497  97.108  97.280  98.548  99.828  99.969  99.984
98.686  98.660  98.396  95.757  93.172  95.752  98.390  98.644  98.680  99.808  99.933  99.948  00.000  00.000  00.000  97.351  97.464  97.488
97.398  97.332  96.873  94.419  92.002  94.421  96.879  97.437  98.585  99.914  99.946  99.950  00.000  00.000  00.000  94.901  94.994  95.022
96.119  96.007  95.404  93.090  90.880  93.067  95.478  96.731  98.217  99.758  99.928  99.948  00.000  00.000  00.000  95.731  94.484  93.834
94.852  94.702  94.078  92.751  91.363  91.926  93.056  94.312  95.848  98.219  98.605  98.775  99.947  99.949  99.948  98.358  96.865  96.200
93.597  93.415  92.784  91.576  90.377  90.701  90.789  92.019  94.296  96.727  97.393  98.605  99.928  99.946  99.933  98.610  97.485  97.383
92.355  92.148  91.516  90.393  89.341  89.424  88.733  90.656  92.926  95.343  96.726  98.219  99.758  99.914  99.804  98.651  98.639  98.650
91.125  90.899  90.273  89.229  88.297  88.342  89.058  90.399  91.706  92.926  94.296  95.848  98.217  98.581  98.645  99.804  99.932  99.948
89.909  89.670  89.054  88.084  87.253  87.283  88.058  89.223  90.399  90.656  92.017  94.308  96.724  97.371  98.581  99.913  99.946  99.949
88.707  88.458  87.856  86.955  86.210  86.239  87.036  88.057  89.053  88.677  90.812  93.071  95.354  96.723  98.216  99.757  99.928  99.947
87.521  87.274  86.773  00.000  00.000  00.000  00.000  00.000  00.000  00.000  00.000  00.000  93.071  94.307  95.847  98.217  98.592  98.644
86.350  86.113  85.693  00.000  00.000  00.000  00.000  00.000  00.000  00.000  00.000  00.000  90.812  92.016  94.285  96.710  97.257  97.353
85.196  84.973  84.618  00.000  00.000  00.000  00.000  00.000  00.000  00.000  00.000  00.000  88.758  90.752  92.943  95.253  95.928  96.071
84.057  83.838  83.415  82.199  81.024  81.536  82.656  83.795  84.955  86.136  87.340  88.567  89.818  91.229  92.612  93.936  94.620  94.801
82.931  82.712  82.250  81.178  80.214  81.219  82.296  83.388  84.494  85.615  86.751  87.904  89.073  90.261  91.452  92.651  93.332  93.543
```

Table 6. Value Iteration results for scale = 3

Value Iteration

Scale = 3

Discount Factor = 0.990

Utility Upper Bound = 100.00

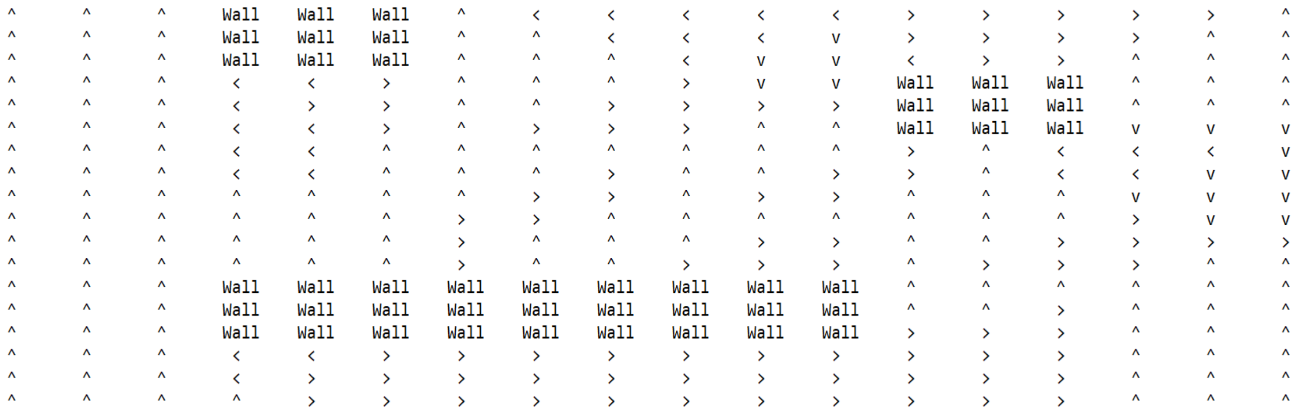
Max Reward (Rmax) = 1.0

Constant 'c' = 0.1

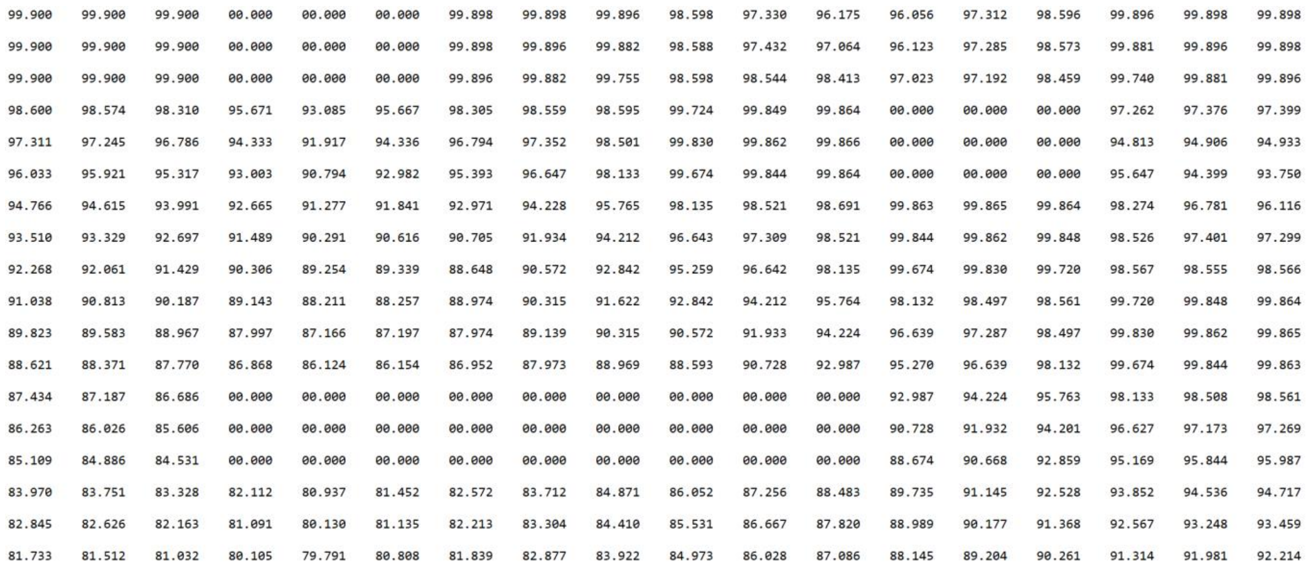
Epsilon Value (c * Rmax) = 0.1

Convergence Threshold = 0.00101

Iterations = 688



```
^     ^     ^     Wall  Wall  Wall  ^     <     <     <     <     <     >     >     >     >     >     ^
^     ^     ^     Wall  Wall  Wall  ^     ^     <     <     <     v     >     >     >     >     ^     ^
^     ^     ^     Wall  Wall  Wall  ^     ^     ^     <     v     v     <     >     >     ^     ^     ^
^     ^     ^     <     <     >     ^     ^     ^     >     v     v     Wall  Wall  Wall  ^     ^     ^
^     ^     ^     <     >     >     ^     ^     >     >     >     >     Wall  Wall  Wall  ^     ^     ^
^     ^     ^     <     <     >     ^     >     >     >     ^     ^     Wall  Wall  Wall  v     v     v
^     ^     ^     <     <     ^     ^     ^     ^     ^     ^     ^     >     ^     <     <     <     v
^     ^     ^     <     <     ^     ^     ^     >     ^     ^     >     >     ^     <     <     v     v
^     ^     ^     ^     ^     ^     ^     >     >     ^     >     >     ^     ^     ^     v     v     v
^     ^     ^     ^     ^     ^     >     >     ^     ^     ^     ^     ^     ^     ^     >     v     v
^     ^     ^     ^     ^     ^     >     ^     ^     ^     >     >     ^     ^     >     >     >     >
^     ^     ^     ^     ^     ^     >     ^     ^     >     >     >     ^     >     >     >     ^     ^
^     ^     ^     Wall  Wall  Wall  Wall  Wall  Wall  Wall  Wall  Wall  ^     ^     ^     ^     ^     ^
^     ^     ^     Wall  Wall  Wall  Wall  Wall  Wall  Wall  Wall  Wall  ^     ^     >     ^     ^     ^
^     ^     ^     Wall  Wall  Wall  Wall  Wall  Wall  Wall  Wall  Wall  >     >     >     ^     ^     ^
^     ^     ^     <     <     >     >     >     >     >     >     >     >     >     >     ^     ^     ^
^     ^     ^     <     >     >     >     >     >     >     >     >     >     >     >     ^     ^     ^
^     ^     ^     ^     >     >     >     >     >     >     >     >     >     >     >     ^     ^     ^
```

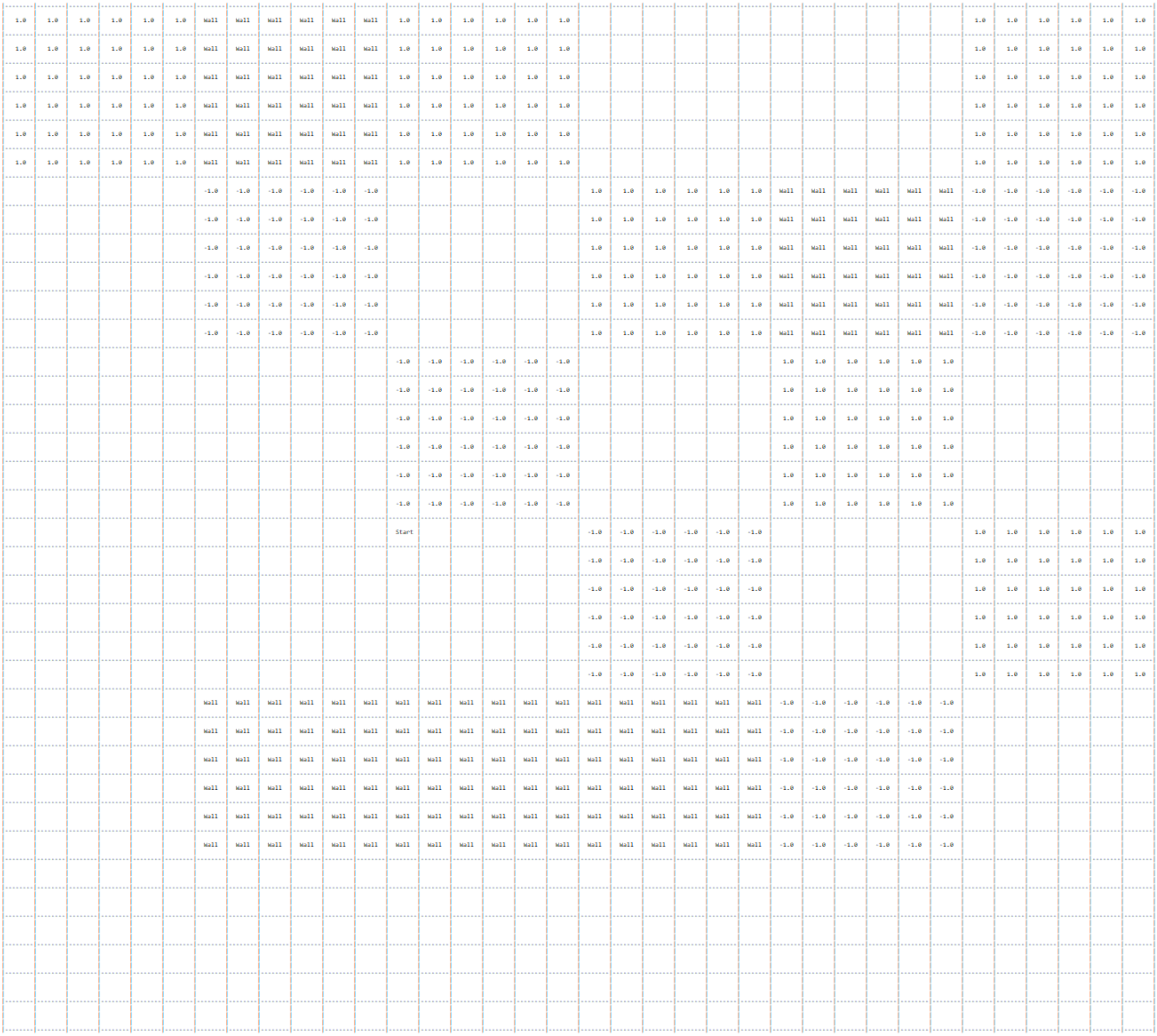


```
99.900  99.900  99.900  00.000  00.000  00.000  99.898  99.898  99.896  98.598  97.330  96.175  96.056  97.312  98.596  99.896  99.898  99.898
99.900  99.900  99.900  00.000  00.000  00.000  99.898  99.896  99.882  98.588  97.432  97.064  96.123  97.285  98.573  99.881  99.896  99.898
99.900  99.900  99.900  00.000  00.000  00.000  99.896  99.882  99.755  98.598  98.544  98.413  97.023  97.192  98.459  99.740  99.881  99.896
98.600  98.574  98.310  95.671  93.085  95.667  98.305  98.559  98.595  99.724  99.849  99.864  00.000  00.000  00.000  97.262  97.376  97.399
97.311  97.245  96.786  94.333  91.917  94.336  96.794  97.352  98.501  99.830  99.862  99.866  00.000  00.000  00.000  94.813  94.906  94.933
96.033  95.921  95.317  93.003  90.794  92.982  95.393  96.647  98.133  99.674  99.844  99.864  00.000  00.000  00.000  95.647  94.399  93.750
94.766  94.615  93.991  92.665  91.277  91.841  92.971  94.228  95.765  98.135  98.521  98.691  99.863  99.865  99.864  98.274  96.781  96.116
93.510  93.329  92.697  91.489  90.291  90.616  90.705  91.934  94.212  96.643  97.309  98.521  99.844  99.862  99.848  98.526  97.401  97.299
92.268  92.061  91.429  90.306  89.254  89.339  88.648  90.572  92.842  95.259  96.642  98.135  99.674  99.830  99.720  98.567  98.555  98.566
91.038  90.813  90.187  89.143  88.211  88.257  88.974  90.315  91.622  92.842  94.212  95.764  98.132  98.497  98.561  99.720  99.848  99.864
89.823  89.583  88.967  87.997  87.166  87.197  87.974  89.139  90.315  90.572  91.933  94.224  96.639  97.287  98.497  99.830  99.862  99.865
88.621  88.371  87.770  86.868  86.124  86.154  86.952  87.973  88.969  88.593  90.728  92.987  95.270  96.639  98.132  99.674  99.844  99.863
87.434  87.187  86.686  00.000  00.000  00.000  00.000  00.000  00.000  00.000  00.000  00.000  92.987  94.224  95.763  98.133  98.508  98.561
86.263  86.026  85.606  00.000  00.000  00.000  00.000  00.000  00.000  00.000  00.000  00.000  90.728  91.932  94.201  96.627  97.173  97.269
85.109  84.886  84.531  00.000  00.000  00.000  00.000  00.000  00.000  00.000  00.000  00.000  88.674  90.668  92.859  95.169  95.844  95.987
83.970  83.751  83.328  82.112  80.937  81.452  82.572  83.712  84.871  86.052  87.256  88.483  89.735  91.145  92.528  93.852  94.536  94.717
82.845  82.626  82.163  81.091  80.130  81.135  82.213  83.304  84.410  85.531  86.667  87.820  88.989  90.177  91.368  92.567  93.248  93.459
81.733  81.512  81.032  80.105  79.791  80.808  81.839  82.877  83.922  84.973  86.028  87.086  88.145  89.204  90.261  91.314  91.981  92.214
```

The policies obtained from both the algorithms are similar. However, there are some differences. Regardless of these differences, they do not follow the optimal policy as it was found earlier with the grid environment given in the question. In that case, the agent made its way to the (0,0) state and stayed there to get +1.0 reward.

Since the number of states is considerably larger than in the question, using a higher discount factor can help in assigning more priority to future rewards. Another point to note is that increasing the discount factor will decrease the convergence threshold if we use other variables (c) with the same values. This will result in an increase in the number of iterations causing heap memory to run out.

When scale = 6, the grid environment is initialized as follows:



1.0	1.0	1.0	1.0	1.0	1.0	Wall	Wall	Wall	Wall	Wall	Wall	1.0	1.0	1.0	1.0	1.0	1.0													1.0	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	Wall	Wall	Wall	Wall	Wall	Wall	1.0	1.0	1.0	1.0	1.0	1.0													1.0	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	Wall	Wall	Wall	Wall	Wall	Wall	1.0	1.0	1.0	1.0	1.0	1.0													1.0	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	Wall	Wall	Wall	Wall	Wall	Wall	1.0	1.0	1.0	1.0	1.0	1.0													1.0	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	Wall	Wall	Wall	Wall	Wall	Wall	1.0	1.0	1.0	1.0	1.0	1.0													1.0	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	Wall	Wall	Wall	Wall	Wall	Wall	1.0	1.0	1.0	1.0	1.0	1.0													1.0	1.0	1.0	1.0	1.0	1.0
						-1.0	-1.0	-1.0	-1.0	-1.0	-1.0							1.0	1.0	1.0	1.0	1.0	1.0	Wall	Wall	Wall	Wall	Wall	Wall	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0
						-1.0	-1.0	-1.0	-1.0	-1.0	-1.0							1.0	1.0	1.0	1.0	1.0	1.0	Wall	Wall	Wall	Wall	Wall	Wall	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0
						-1.0	-1.0	-1.0	-1.0	-1.0	-1.0							1.0	1.0	1.0	1.0	1.0	1.0	Wall	Wall	Wall	Wall	Wall	Wall	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0
						-1.0	-1.0	-1.0	-1.0	-1.0	-1.0							1.0	1.0	1.0	1.0	1.0	1.0	Wall	Wall	Wall	Wall	Wall	Wall	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0
						-1.0	-1.0	-1.0	-1.0	-1.0	-1.0							1.0	1.0	1.0	1.0	1.0	1.0	Wall	Wall	Wall	Wall	Wall	Wall	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0
						-1.0	-1.0	-1.0	-1.0	-1.0	-1.0							1.0	1.0	1.0	1.0	1.0	1.0	Wall	Wall	Wall	Wall	Wall	Wall	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0
												-1.0	-1.0	-1.0	-1.0	-1.0	-1.0							1.0	1.0	1.0	1.0	1.0	1.0						
												-1.0	-1.0	-1.0	-1.0	-1.0	-1.0							1.0	1.0	1.0	1.0	1.0	1.0						
												-1.0	-1.0	-1.0	-1.0	-1.0	-1.0							1.0	1.0	1.0	1.0	1.0	1.0						
												-1.0	-1.0	-1.0	-1.0	-1.0	-1.0							1.0	1.0	1.0	1.0	1.0	1.0						
												-1.0	-1.0	-1.0	-1.0	-1.0	-1.0							1.0	1.0	1.0	1.0	1.0	1.0						
												-1.0	-1.0	-1.0	-1.0	-1.0	-1.0							1.0	1.0	1.0	1.0	1.0	1.0						
												Start						-1.0	-1.0	-1.0	-1.0	-1.0	-1.0							1.0	1.0	1.0	1.0	1.0	1.0
																		-1.0	-1.0	-1.0	-1.0	-1.0	-1.0							1.0	1.0	1.0	1.0	1.0	1.0
																		-1.0	-1.0	-1.0	-1.0	-1.0	-1.0							1.0	1.0	1.0	1.0	1.0	1.0
																		-1.0	-1.0	-1.0	-1.0	-1.0	-1.0							1.0	1.0	1.0	1.0	1.0	1.0
																		-1.0	-1.0	-1.0	-1.0	-1.0	-1.0							1.0	1.0	1.0	1.0	1.0	1.0
																		-1.0	-1.0	-1.0	-1.0	-1.0	-1.0							1.0	1.0	1.0	1.0	1.0	1.0
						Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0						
						Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0						
						Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0						
						Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0						
						Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0						
						Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	Wall	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0						

Figure 15. Complex grid model with scale = 6

In the above figure, the cells with +1.0 rewards are green squares, -1.0 are brown squares and white squares have -0.4 reward.

Policy iteration and value iteration is executed in the similar way as done earlier. The results for which are discussed as follows:

Table 7. Configuration for grid model with scale = 6

Policy Iteration	Value Iteration

Scale = 6
Discount Factor = 0.99
K = 50
	Scale = 6
	Discount Factor = 0.99
	Utility Upper Bound = 100.00
	Max Reward (Rmax) = 1.0
	Constant 'c' = 0.1
	Epsilon Value (c * Rmax) = 0.1
	Convergence Threshold = 0.00101

The optimal policy and the utility values can be found in the text files “CZ4046-Assignment1\GridWorld\results\complex grid results\config_results_value_6.txt” and “CZ4046-Assignment1\GridWorld\results\complex grid results\value_iteration_utilities_scale_6.csv” (for value iteration) and “CZ4046-Assignment1\GridWorld\results\complex grid results\config_results_policy_6.txt” and “CZ4046-Assignment1\GridWorld\results\complex grid results\policy_iteration_utilities_scale_6.csv” (for policy iteration).

Both the algorithms are not able to learn the optimal policy. This maybe because the number of states has been increased sufficiently to 36*36. Therefore, the convergence is reached before the optimal policy can be learned. This can be avoided using a higher discount factor, which would enable the algorithm to run for more iterations.

8. Appendix