ARTICIAL INTELLIGENCE
ASSIGNMENT 4 REPORT

RANDOM MAZE SOLVER

USING REINFORCEMENT LEARNING

STUDENTS:

HASSAN TAMER 7405
MAZEN GABER 7467
OMAR ESSAM 7859

TABLE OF CONTENTS

Game Description

The Random Maze Solver is a game that generates a random maze based on the size of maze as an input from the user, then the user chooses which solving algorithm technique to apply in order to solve the maze.

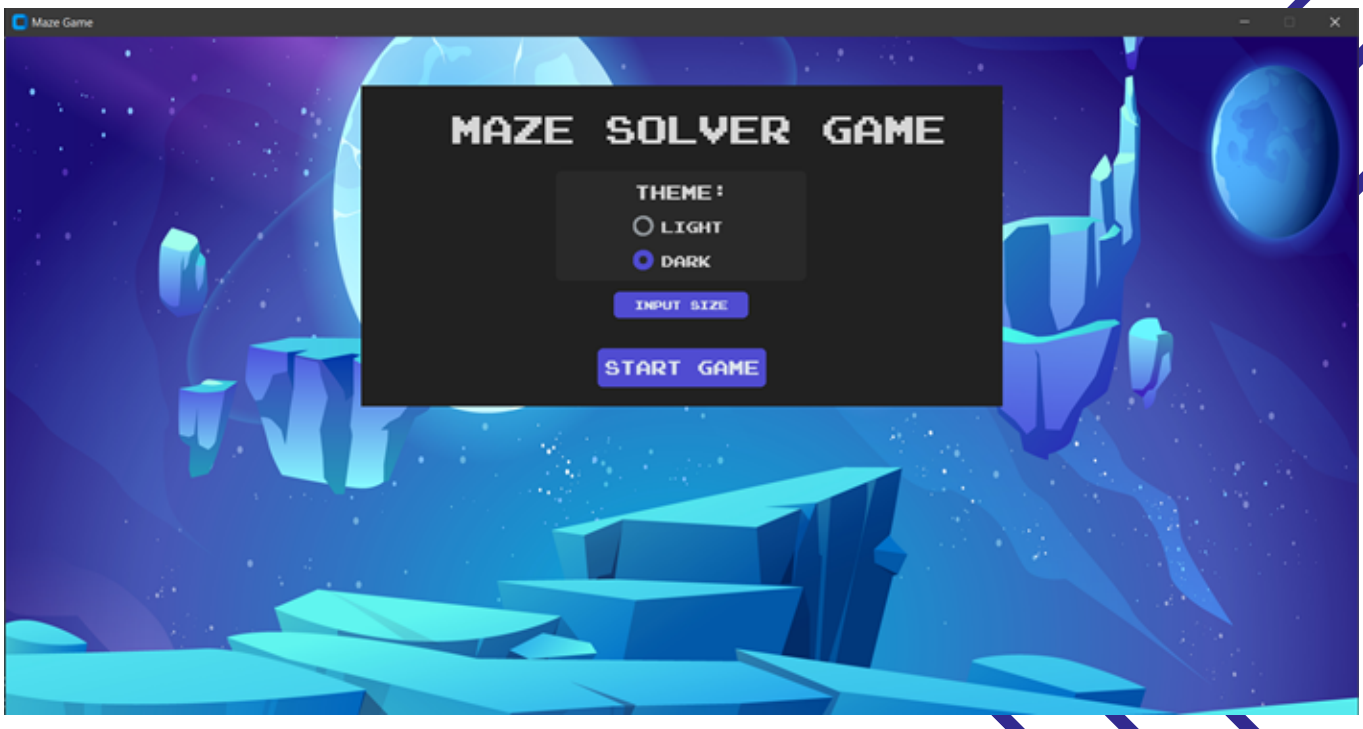
In order to run the game, please run the file `maze_GUI.py`

GUI AND USER GUIDE

THE GAME ASSUMES THAT THE USER IS FAMILIAR WITH THE GUI BASED APPLICATIONS AND FACILITATES TO THE USER THE USAGE OF THE GAME THROUGH SIMPLE INSTRUCTIONS:

1. MAIN START PAGE

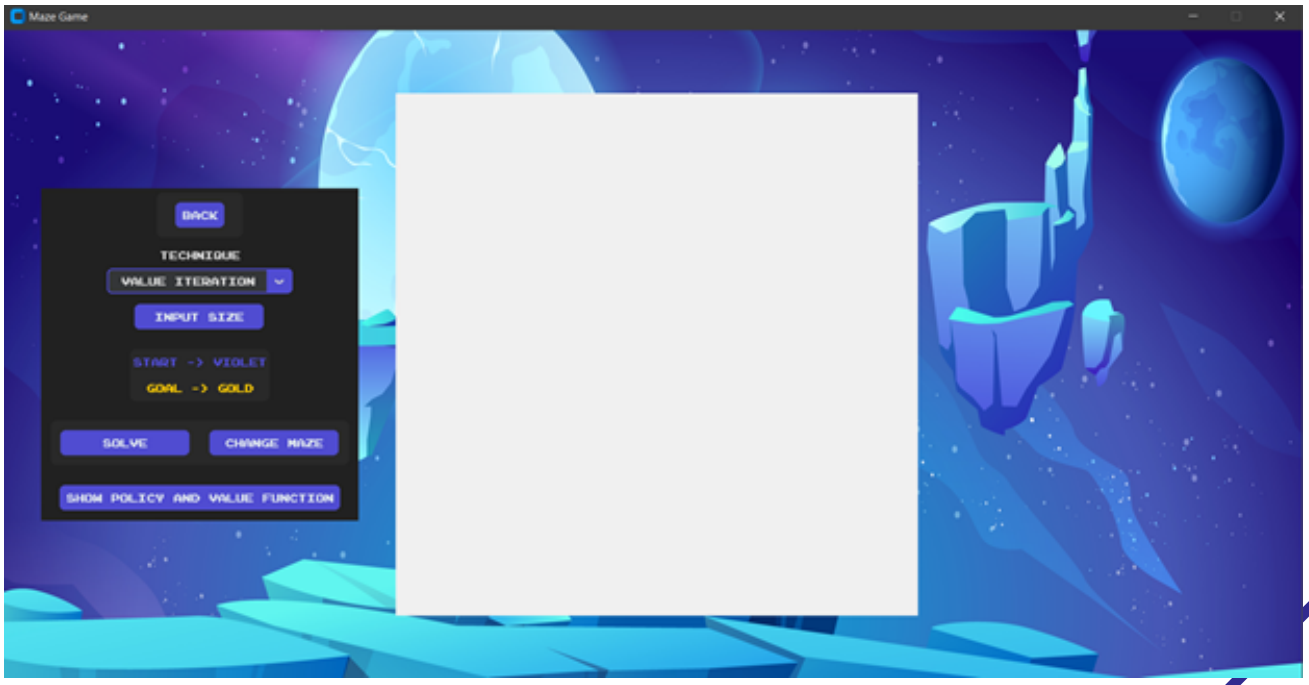
In this page, the user can input the size of the maze board desired through the INPUT SIZE button. Then, the user presses start game.

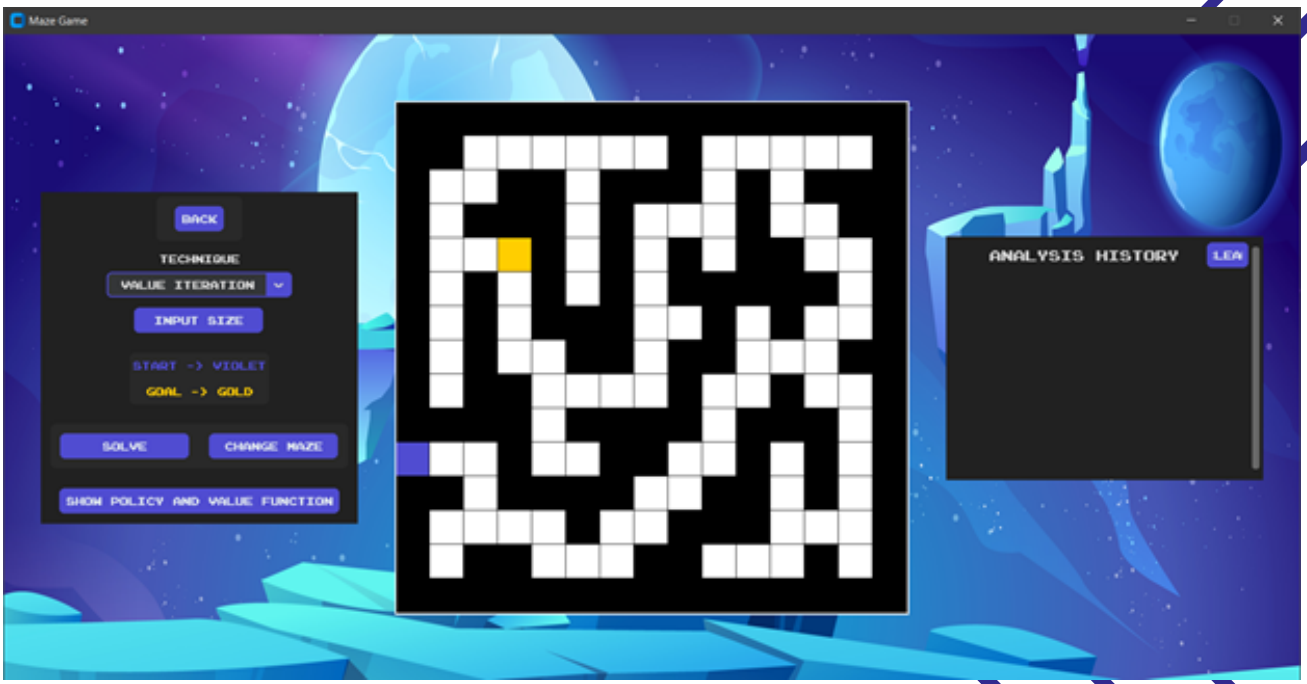


2. GAME PAGE

In this page, the user can alter the maze generated and apply the solving technique preferred.

NOTE: If the maze board appeared as a white board as the following. please press change maze to reconstruct a new maze. this is because of the random generated maze might have an issue with rendering to the canvas. (The second image is fully rendered)





ON THE LEFT: *THE CONTROL MENU*:

This is the menu where the user is able to choose the technique to use {value iteration, policy iteration}. The user can also change the size of the maze by clicking on the input size button. Then by pressing the solve button. the game will be automatically solved; Displaying in the Analysis History the time taken and the cost of the optimal path. Afterwards, clicking on the "show policy and value function" button will redirect the user to another frame where the optimal policy and the optimal value functions are shown.

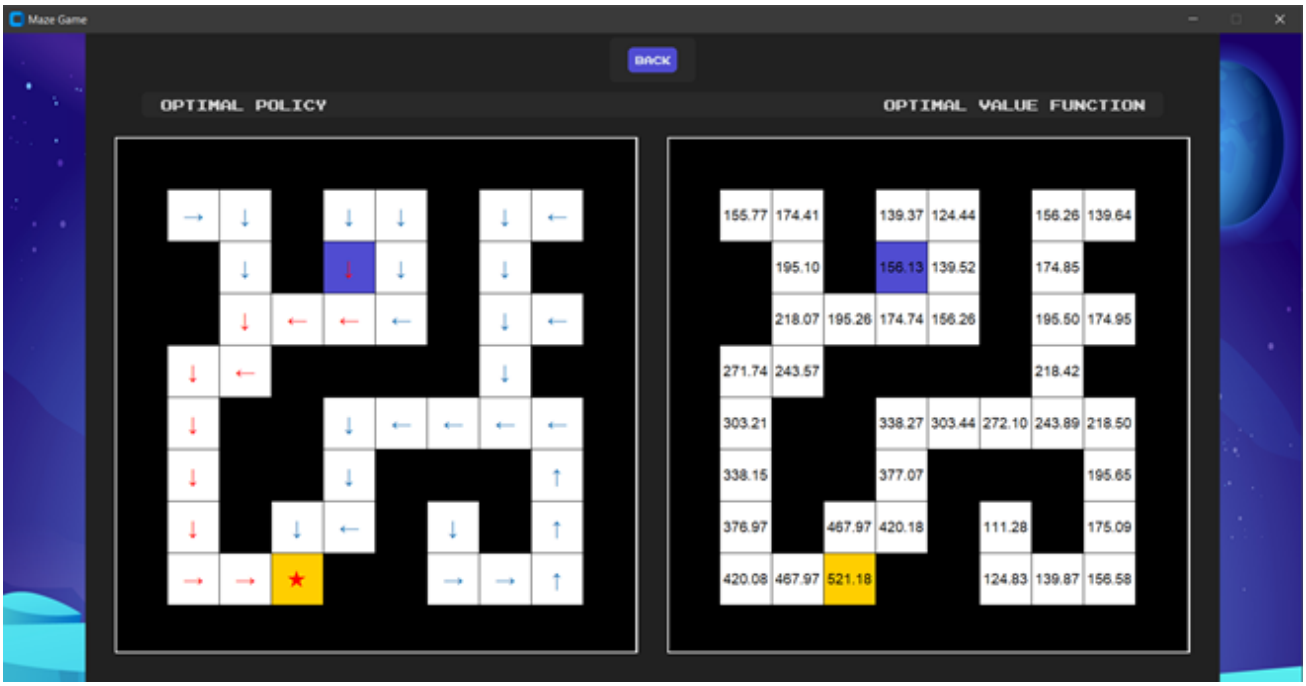
IN THE MIDDLE: *THE MAZE BOARD*:

ON THE RIGHT: *THE ANALYSIS MENU*:

This is where the result analysis will be shown such as the time taken, the and the cost.

2. RESULT PAGE

In this page, the optimal policy and the optimal value functions are displayed on two different boards. in case if the user chose the policy iteration algorithm, then the optimal policy will be shown consequently.
in the optimal policy, the optimal path is shown by the red arrows. (*Illustrating image in the next page*).



NOTE: **If the value functions are not readable because the input size might be too big, please return to the console log output where the necessary utilities for tracing are printed such as the number of the iterations, the output after each iteration and so on**

```
Optimal Value Function:
[[  0.           0.           0.           0.           0.
    0.           0.           0.           0.           0.        ]
 [  0.         155.77368108 174.41283542   0.         139.37439669
  124.43695702   0.         156.26165026 139.63548523   0.        ]
 [  0.           0.         195.10102127   0.         156.13182722
  139.5186445    0.         174.85200768   0.           0.        ]
 [  0.           0.         218.06810735 195.26129661 174.73516695
  156.26165026   0.         195.4962763  174.94664867   0.        ]
 [  0.         271.74364835 243.56928351   0.           0.
    0.           0.         218.42383688   0.           0.        ]
 [  0.         303.20877351   0.           0.         338.27083689
  303.4437532  272.09937788 243.88944009 218.50049608   0.        ]
 [  0.         338.15399616   0.           0.         377.0727532
    0.           0.           0.         195.65044648   0.        ]
 [  0.         376.96759655   0.         467.9727532  420.17547788
    0.         111.28140224   0.         175.08540183   0.        ]
 [  0.         420.08083689 467.9727532  521.17547788   0.
    0.         124.82610688 139.86887938 156.57686165   0.        ]
 [  0.           0.           0.           0.           0.
    0.           0.           0.           0.           0.        ]]

Optimal Policy:
[['X' 'X' 'X' 'X' 'X' 'X' 'X' 'X' 'X' 'X']
 ['X' 'R' 'D' 'X' 'D' 'D' 'X' 'D' 'L' 'X']
 ['X' 'X' 'D' 'X' 'D' 'D' 'X' 'D' 'X' 'X']
 ['X' 'X' 'D' 'L' 'L' 'L' 'X' 'D' 'L' 'X']
 ['X' 'D' 'L' 'X' 'X' 'X' 'X' 'D' 'X' 'X']
 ['X' 'D' 'X' 'X' 'D' 'L' 'L' 'L' 'L' 'X']
 ['X' 'D' 'X' 'X' 'D' 'X' 'X' 'X' 'U' 'X']
 ['X' 'D' 'X' 'D' 'L' 'X' 'D' 'X' 'U' 'X']
 ['X' 'R' 'R' 'S' 'X' 'X' 'R' 'R' 'U' 'X']
 ['X' 'X' 'X' 'X' 'X' 'X' 'X' 'X' 'X' 'X']]
```

Algorithms and Data Structures

Algorithms:

The algorithms used are the same provided in the assignment pdf, no updates where modified but GUI functions to enhance the visualization of the maze and provide a better experience.

Data Structures:

The maze is represented and manipulated using a 2D matrix. With extra data structures for maintaining the GUI variables (out of scope).

Maze example:

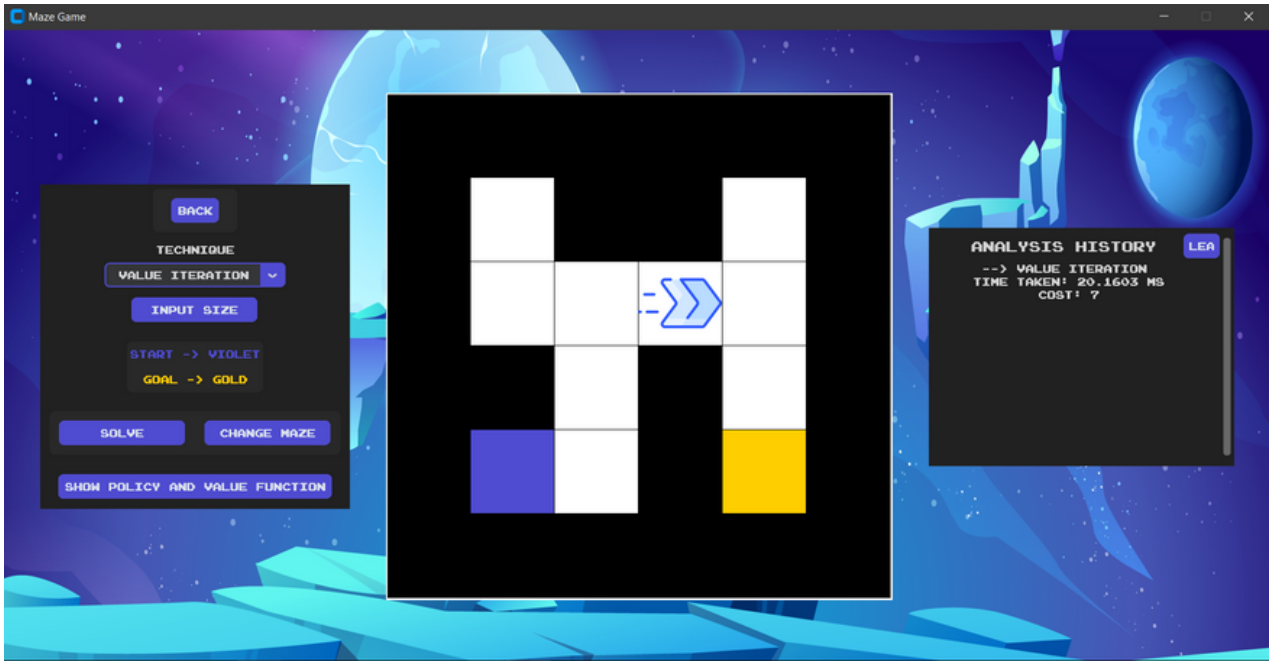
```
Maze ------------
[0, 0, 0, 0, 0, 0]
[0, 1, 0, 0, 1, 0]
[0, 1, 1, 1, 1, 0]
[0, 0, 1, 0, 1, 0]
[0, 2, 1, 0, 3, 0]
[0, 0, 0, 0, 0, 0]
start:  (4, 1)
goal:  (4, 4)
```

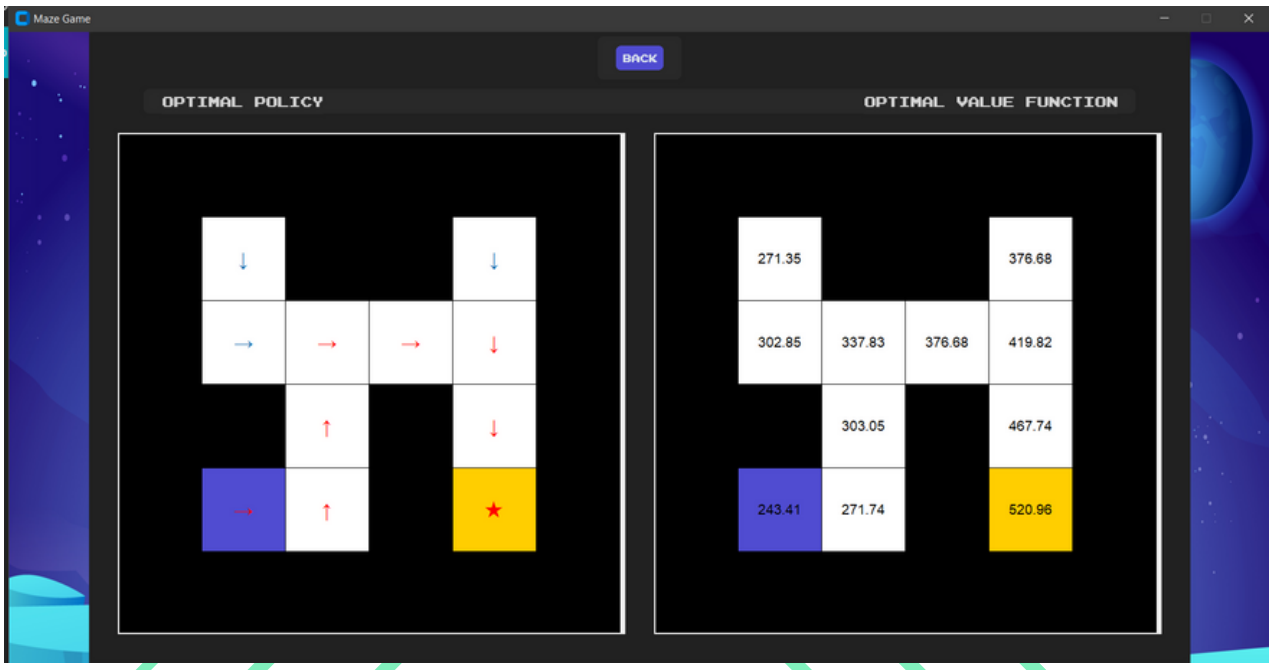
0 -> wall
1 -> empty cell
2 -> start position
3 -> goal position
Rewards:
-1 **per step**
+100 **for goal**

Sample Runs

Please note that the arrow displayed on the maze is a because of a snapshot of the solution animation as it animates the optimal solution from the start position to the goal position. The firework represents reaching the goal state. If N is too large, it might cause rendering obstacles for policy iteration since the customtkinter clock makes issues, if so, please return to the console log to review the analysis.

Run 1 (N = 6):

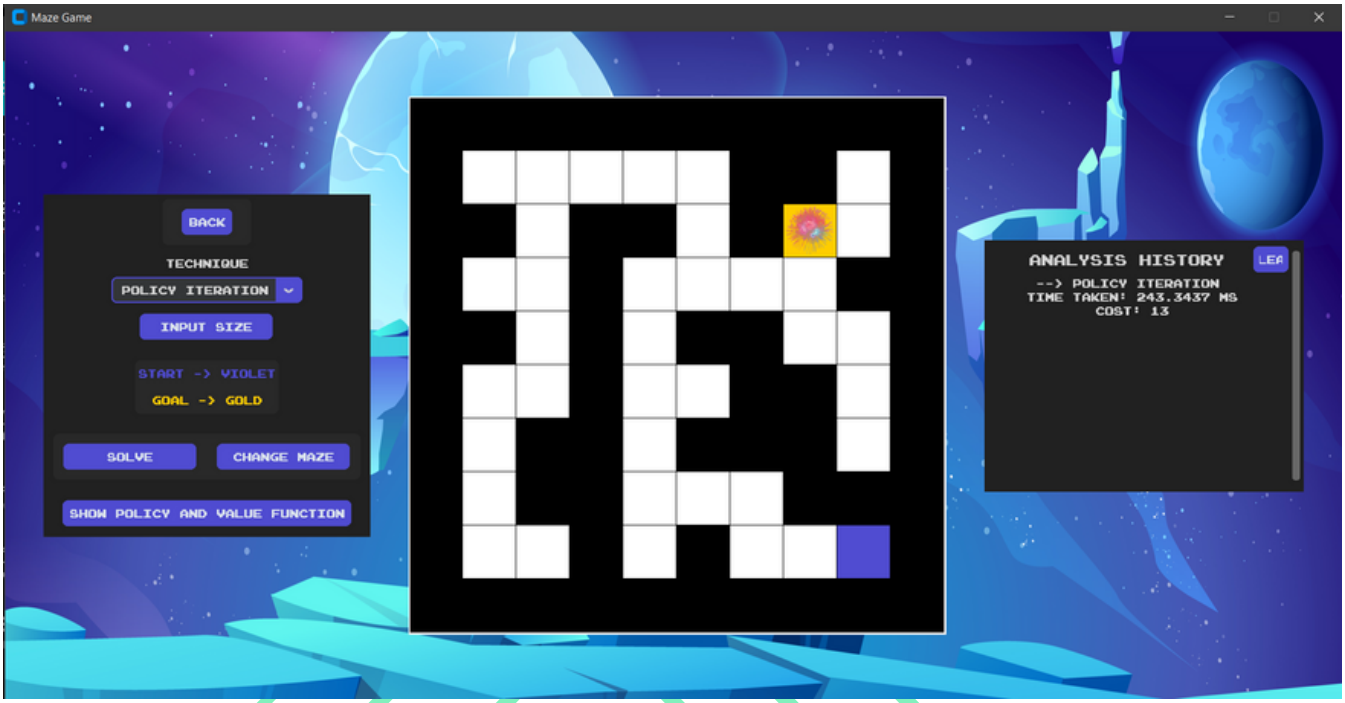


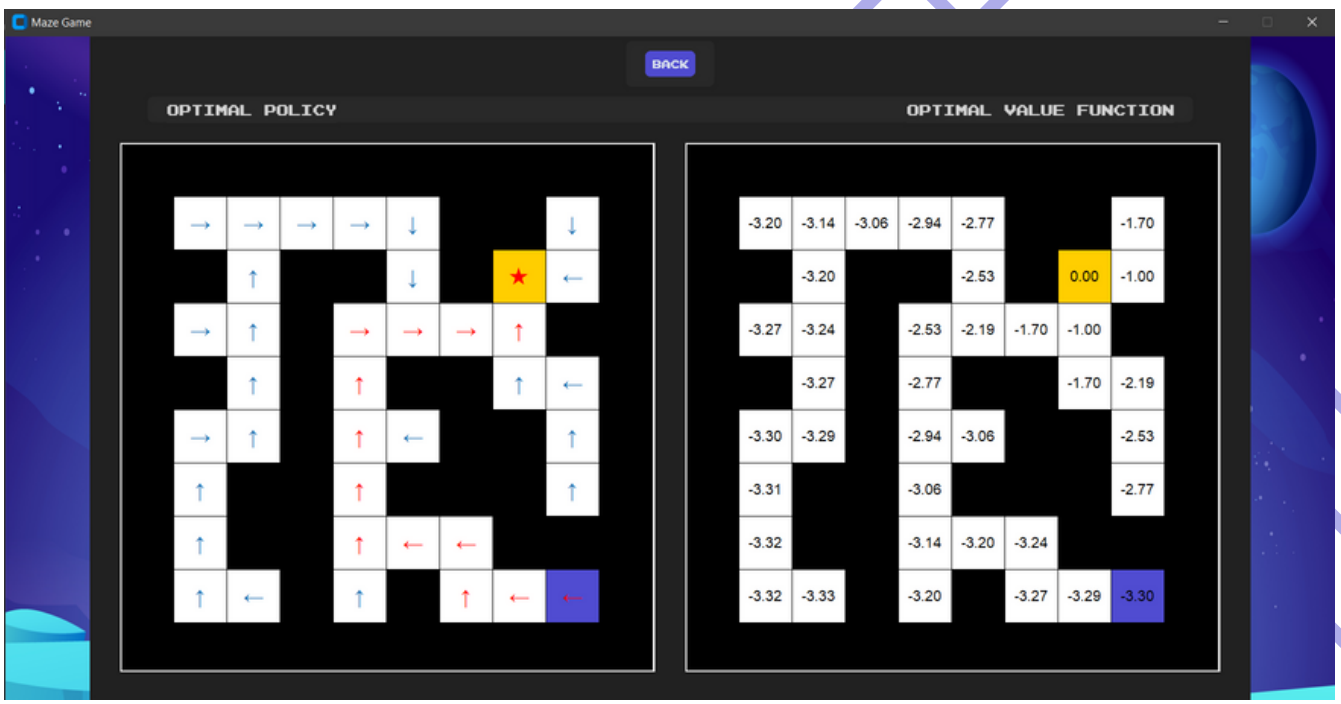


```
**********
Total iterations:  32
**********
4 1
[(4, 1), (4, 2), (3, 2), (2, 2), (2, 3), (2, 4), (3, 4), (4, 4)]
Optimal Value Function:
[[  0.          0.          0.          0.          0.
    0.        ]
 [  0.        271.34570818   0.          0.        376.67749817
    0.        ]
 [  0.        302.85062736 337.83166463 376.67749817 419.81974835
    0.        ]
 [  0.          0.        303.04849817   0.        467.73777351
    0.        ]
 [  0.        243.40900817 271.74364835   0.        520.96399616
    0.        ]
 [  0.          0.          0.          0.          0.
    0.        ]]

Optimal Policy:
[['X' 'X' 'X' 'X' 'X' 'X']
 ['X' 'D' 'X' 'X' 'D' 'X']
 ['X' 'R' 'R' 'R' 'D' 'X']
 ['X' 'X' 'U' 'X' 'D' 'X']
 ['X' 'R' 'U' 'X' 'S' 'X']
 ['X' 'X' 'X' 'X' 'X' 'X']]
pov True
IN VISUALIZE POLICY
[['X' 'X' 'X' 'X' 'X' 'X']
 ['X' 'D' 'X' 'X' 'D' 'X']
 ['X' 'R' 'R' 'R' 'D' 'X']
 ['X' 'X' 'U' 'X' 'D' 'X']
 ['X' 'R' 'U' 'X' 'S' 'X']
 ['X' 'X' 'X' 'X' 'X' 'X']]
maze:
[[0, 0, 0, 0, 0, 0], [0, 1, 0, 0, 1, 0], [0, 1, 1, 1, 1, 0], [0, 0, 1, 0, 1, 0], [0, 2, 1, 0, 3, 0], [0, 0, 0, 0, 0, 0]]
```

Run 2 (N = 10):



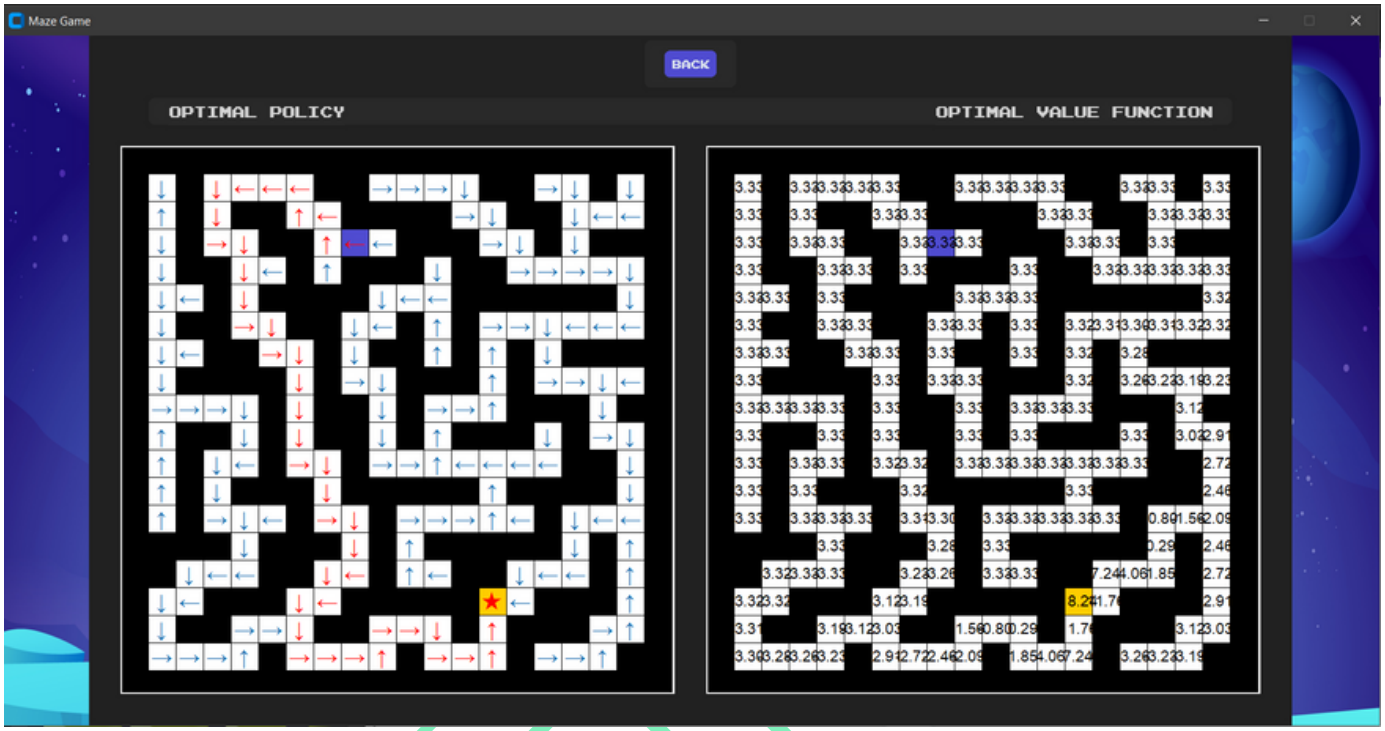



```
Iterations:  10
8 8
[(8, 8), (8, 7), (8, 6), (7, 6), (7, 5), (7, 4), (6, 4), (5, 4), (4, 4), (3, 4), (3, 5), (3, 6), (3, 7), (2, 7)]
Optimal Value Function:
[[ 0.         0.         0.         0.         0.         0.
   0.         0.         0.         0.       ]
 [ 0.        -3.1988213 -3.1411734 -3.058819  -2.94117   -2.7731
   0.         0.        -1.7        0.       ]
 [ 0.         0.        -3.1988213  0.         0.        -2.533
   0.         0.        -1.         0.       ]
 [ 0.        -3.2674224 -3.2391748  0.        -2.533     -2.19
  -1.7       -1.         0.         0.       ]
 [ 0.         0.        -3.2674224  0.        -2.7731    0.
   0.        -1.7       -2.19       0.       ]
 [ 0.        -3.301037  -3.2871957  0.        -2.94117   -3.058819
   0.         0.        -2.533      0.       ]
 [ 0.        -3.310726   0.         0.        -3.058819  0.
   0.         0.        -2.7731     0.       ]
 [ 0.        -3.3175082  0.         0.        -3.1411734 -3.1988213
  -3.2391748  0.         0.         0.       ]
 [ 0.        -3.3222558 -3.3255792  0.        -3.1988213  0.
  -3.2674224 -3.2871957 -3.301037   0.       ]
 [ 0.         0.         0.         0.         0.         0.
   0.         0.         0.         0.       ]]

Optimal Policy:
[['X' 'X' 'X' 'X' 'X' 'X' 'X' 'X' 'X' 'X']
 ['X' 'R' 'R' 'R' 'R' 'D' 'X' 'X' 'D' 'X']
 ['X' 'X' 'U' 'X' 'X' 'D' 'X' 'S' 'L' 'X']
 ['X' 'R' 'U' 'X' 'R' 'R' 'R' 'U' 'X' 'X']
 ['X' 'X' 'U' 'X' 'U' 'X' 'X' 'U' 'L' 'X']
 ['X' 'R' 'U' 'X' 'U' 'L' 'X' 'X' 'U' 'X']
 ['X' 'U' 'X' 'X' 'U' 'X' 'X' 'X' 'U' 'X']
 ['X' 'U' 'X' 'X' 'U' 'L' 'L' 'X' 'X' 'X']
 ['X' 'U' 'L' 'X' 'U' 'X' 'U' 'L' 'L' 'X']
 ['X' 'X' 'X' 'X' 'X' 'X' 'X' 'X' 'X' 'X']]
----------------------------END OF Policy Iteration----------------------------
```

Run 3 (N = 20):


Maze Game
BACK
TECHNIQUE
VALUE ITERATION
INPUT SIZE
START -> VIOLET
GOAL -> GOLD
SOLVE
CHANGE MAZE
SHOW POLICY AND VALUE FUNCTION
ANALYSIS HISTORY
--> VALUE ITERATION
TIME TAKEN: 328.1217 MS
COST: 42


Maze Game
BACK
OPTIMAL POLICY
OPTIMAL VALUE FUNCTION

```
**********
Total iterations:  34
**********
3 8
[(3, 8), (3, 7), (2, 7), (2, 6), (1, 6), (1, 5), (1, 4), (1, 3), (2, 3), (3, 3), (3, 4), (4, 4), (5, 4), (6, 4), (6, 5), (7, 5), (7, 6), (8, 6), (9, 6), (10,
6), (11, 6), (11, 7), (12, 7), (13, 7), (13, 8), (14, 8), (15, 8), (15, 7), (16, 7), (16, 6), (17, 6), (18, 6), (18, 7), (18, 8), (18, 9), (17, 9), (17, 10),
(17, 11), (18, 11), (18, 12), (18, 13), (17, 13), (16, 13)]
Optimal Value Function:
[[ 0.         0.         0.         0.         0.         0.
   0.         0.         0.         0.         0.         0.
   0.         0.         0.         0.         0.         0.
   0.         0.        ]
 [ 0.        -3.3333303  0.        -3.33325912 -3.33328138 -3.33329697
  -3.33330788  0.         0.        -3.33325251 -3.33321723 -3.33316702
  -3.33309543  0.         0.        -3.33263895 -3.33234128  0.
  -3.33284723  0.        ]
 [ 0.        -3.3333303  0.        -3.33322185  0.         0.
  -3.33331551 -3.33332086  0.         0.         0.         0.
  -3.33299325 -3.33284734  0.         0.        -3.33191606 -3.33234124
  -3.33263887  0.        ]
 [ 0.        -3.3333303  0.        -3.33317026 -3.33309769  0.
   0.        -3.3333246  -3.33332722 -3.33332905  0.         0.
   0.        -3.33263895 -3.33234128  0.        -3.33130862  0.
   0.         0.        ]
 [ 0.        -3.33332484  0.         0.        -3.33299483 -3.33309638
   0.        -3.33332722  0.         0.         0.        -3.33331975
   0.         0.        -3.33191606 -3.33130862 -3.33044086 -3.32920121
  -3.32743028  0.        ]
 [ 0.        -3.33331739 -3.33332217  0.        -3.33284845  0.
   0.         0.         0.        -3.33329351 -3.33330546 -3.33331382
   0.         0.         0.         0.         0.         0.
  -3.32490039  0.        ]
 [ 0.        -3.33330788  0.         0.        -3.33263973 -3.33234182
   0.         0.        -3.33325184 -3.33327629  0.        -3.33331967
   0.        -3.31612327 -3.30874751 -3.2982107  -3.30874749 -3.31612325
  -3.32128627  0.        ]
 [ 0.        -3.3332951  -3.33330657  0.         0.        -3.33191644
  -3.33130888  0.        -3.33321676  0.         0.        -3.33332377
```

```
Optimal Policy:
[['X' 'X' 'X' 'X' 'X' 'X' 'X' 'X' 'X' 'X' 'X' 'X' 'X' 'X' 'X' 'X' 'X' 'X'
  'X' 'X']
 ['X' 'D' 'X' 'D' 'L' 'L' 'L' 'X' 'X' 'R' 'R' 'R' 'D' 'X' 'X' 'R' 'D' 'X'
  'D' 'X']
 ['X' 'U' 'X' 'D' 'X' 'X' 'U' 'L' 'X' 'X' 'X' 'X' 'R' 'D' 'X' 'X' 'D' 'L'
  'L' 'X']
 ['X' 'D' 'X' 'R' 'D' 'X' 'X' 'U' 'L' 'L' 'X' 'X' 'X' 'R' 'D' 'X' 'D' 'X'
  'X' 'X']
 ['X' 'D' 'X' 'X' 'D' 'L' 'X' 'U' 'X' 'X' 'X' 'D' 'X' 'X' 'R' 'R' 'R' 'R'
  'D' 'X']
 ['X' 'D' 'L' 'X' 'D' 'X' 'X' 'X' 'X' 'D' 'L' 'L' 'X' 'X' 'X' 'X' 'X' 'X'
  'D' 'X']
 ['X' 'D' 'X' 'X' 'R' 'D' 'X' 'X' 'D' 'L' 'X' 'U' 'X' 'R' 'R' 'D' 'L' 'L'
  'L' 'X']
 ['X' 'D' 'L' 'X' 'X' 'R' 'D' 'X' 'D' 'X' 'X' 'U' 'X' 'U' 'X' 'D' 'X' 'X'
  'X' 'X']
 ['X' 'D' 'X' 'X' 'X' 'X' 'D' 'X' 'R' 'D' 'X' 'X' 'X' 'U' 'X' 'R' 'R' 'D'
  'L' 'X']
 ['X' 'R' 'R' 'R' 'D' 'X' 'D' 'X' 'X' 'D' 'X' 'R' 'R' 'U' 'X' 'X' 'X' 'D'
  'X' 'X']
 ['X' 'U' 'X' 'X' 'D' 'X' 'D' 'X' 'X' 'D' 'X' 'U' 'X' 'X' 'X' 'D' 'X' 'R'
  'D' 'X']
 ['X' 'U' 'X' 'D' 'L' 'X' 'R' 'D' 'X' 'R' 'R' 'U' 'L' 'L' 'L' 'L' 'X' 'X'
  'D' 'X']
 ['X' 'U' 'X' 'D' 'X' 'X' 'X' 'D' 'X' 'X' 'X' 'X' 'X' 'U' 'X' 'X' 'X' 'X'
  'D' 'X']
 ['X' 'U' 'X' 'R' 'D' 'L' 'X' 'R' 'D' 'X' 'R' 'R' 'R' 'U' 'L' 'X' 'D' 'L'
  'L' 'X']
 ['X' 'X' 'X' 'X' 'D' 'X' 'X' 'X' 'D' 'X' 'U' 'X' 'X' 'X' 'X' 'X' 'D' 'X'
  'U' 'X']
 ['X' 'X' 'D' 'L' 'L' 'X' 'X' 'D' 'L' 'X' 'U' 'L' 'X' 'X' 'D' 'L' 'L' 'X'
  'U' 'X']
 ['X' 'D' 'L' 'X' 'X' 'X' 'D' 'L' 'X' 'X' 'X' 'X' 'X' 'S' 'L' 'X' 'X' 'X'
  'U' 'X']
 ['X' 'D' 'X' 'X' 'R' 'R' 'D' 'X' 'X' 'R' 'R' 'D' 'X' 'U' 'X' 'X' 'X' 'R'
  'U' 'X']
```

Result Analysis and Comparisons

It was observed that the *value iteration* algorithm takes in total more iterations to reach convergence and hence provide the optimal convergence. But it is faster in execution than the *policy iteration* since the policy iteration algorithm is divided into 2 partitions: policy evaluation and then policy improvement and once policy evaluation is complete, policy improvement is performed by greedily selecting actions that maximize the expected cumulative rewards based on the updated value function. This step typically requires additional iterations to converge to an optimal policy.

Running time comparison:

N	Value Iteration (ms)	Policy Iteration (ms)
6	20.16	24.23
10	158.9	243.34
20	328.12	762.49