CSE583/EE552 Pattern Recognition and Machine Learning: Project #4 Part 1

Due on March 28th, 2022 at 11:59pm

PROFESSOR Yanxi Liu Spring 22

Anish Phule **asp5607@psu.edu**

This report consists of:

Problem 1 - Reinforcement Learning:

1. General Approach to the problem

2. Description of problems faced

3. Results and figures

4. Extra Credit

Problem 1

Deep Learning

1. General Approach to the problem

The problem was solved in two parts, manual maze solving and learning based solving.

Manual based solving:

- In the manual based solving problem, we first need to analyze which actions the user can take at a certain state. For this, we give each state all the four actions, then analyze the neighborhood of that state. If any of the neighborhood has 1(wall) in the maze, we remove that action.

```
if maze(i,j-1) == 1
    allowed_states(make_coordinates(i,j)) = setdiff(allowed_states(make_coordinates(i,j)), {'L'});
end
if maze(i,j+1) == 1
    allowed_states(make_coordinates(i,j)) = setdiff(allowed_states(make_coordinates(i,j)), {'R'});
end
if maze(i-1,j) == 1
    allowed_states(make_coordinates(i,j)) = setdiff(allowed_states(make_coordinates(i,j)), {'U'});
end
if maze(i+1,j) == 1
    allowed_states(make_coordinates(i,j)) = setdiff(allowed_states(make_coordinates(i,j)), {'D'});
end
```

Figure 1: Algorithm to determine valid action at each state.

- The above gives us valid actions at each state that can be accessed while solving. We then ask the user for input, and cross validate with the available states. If it is not a valid move, we display an error. If it is a valid move, we move the robot to the new position.
- At every point, we keep track of a variable 'tot', which is a product of the state indices. As long as tot is less than 64(8*8), the robot keeps moving. As soon as tot==64, that means the robot has reached the end, and the maze has been completed.

Q-Learning based solving:

- For the Q learning based solving, the first step is the same, wherein we determine what valid actions can be taken by the robot.
- We want out robot to learn, so we need to initialize some rewards for it to pursue. We set the rewards for (0,0) and (8,8) i.e. start and end as 0, and the rest of the maze states have reward -1.
- We also then set the Q Table with expectations for each and every state, that are generated randomly between 0 and 1.
- As the robot moves through the maze, we record its movement as state history. At the end, we go back through the state history, and update current expectation for that state through the following algorithm:
  for length(state table) in reverse:
  current expectation = current expectation + α(total reward - current expectation)
  target reward = target reward + reward(at that state)
  This helps us update the expectations at each state.

Problem 1 continued on next page...

- The reason of updating expectations, is to help our robot learn to explore and exploit the maze. We set a random factor = 0.25, and generate a random number r as well. If r is less than 0.25, it explores the maze, else it exploits. We also decrease the random factor at some interval, thus encouraging the robot to do more exploitation.
- Exploration is choosing a random valid move at a state, exploitation is analyzing the expectations of surrounding actions, and choosing the one with highest expectation.
- As the random factor goes down as well as the number of iterations increase, the expectations get updated, paving the way for the best possible path for the robot to learn. After a few thousand iterations, the robot does only exploitation and only chooses the best path.

2. Description of problems faced

- Setting actions for every state: Because I chose to do this project in matlab, it is not easy to add multiple values to keys in map containers. Although it is possible, I had problems where in some places it didn't save any directions at all(as matlab doesn't have a 'continue' expression for if else, and I couldn't continue where an action was a wall). I ended up removing directions that were walls, instead of adding actions that were not walls.
- Residual values:To visualize the maze, we give the robot's current location the value 2, and update the old location to 0. However, in multiple iterations, it had some residual places where some locations were also showing the value 2. On inspection, I found out because we are changing the maze every iteration(old location = current location), the next time it starts iterating, it isn't exactly the same maze. For this, I duplicated the maze and in every iteration, I do maze = maze duplicate.
- While loop condition:The original starter code had the while loop, wherein it said - while current location $\neq$ end location. But, this in matlab only evaluates one indice. Therefore, even if the rest of the algorithm is correct, the while loop would complete once the robot reached the 8th row. For this, I made a variable 'tot', which is a product of the state indices. So the new condition now becomes, while tot $\neq$ 64, which is unique only to the end location.

3. Results and figures(with explanation)

Please note the following:
random factor = 0.25, with decrement of 0.0001 happening every 500 iterations. alpha = 0.01
Total 5000 iterations happen, and heatmaps are generated every 250 iterations.
Also note that since the (0,0) is start position, the heatmaps don't show the first position.

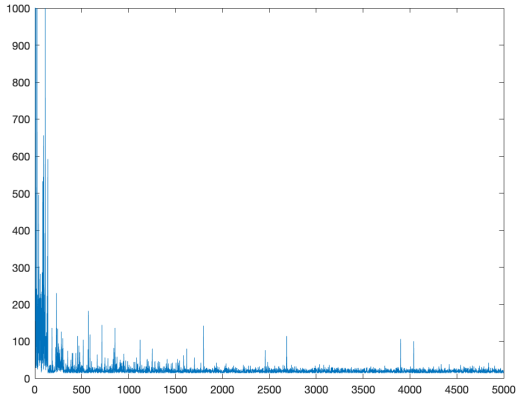


Figure 2: Number of moves taken by robot per iteration

The robot takes a large number of moves in the start as it explores as seen in fig 2, but as it learns the number of moves goes on decreasing.

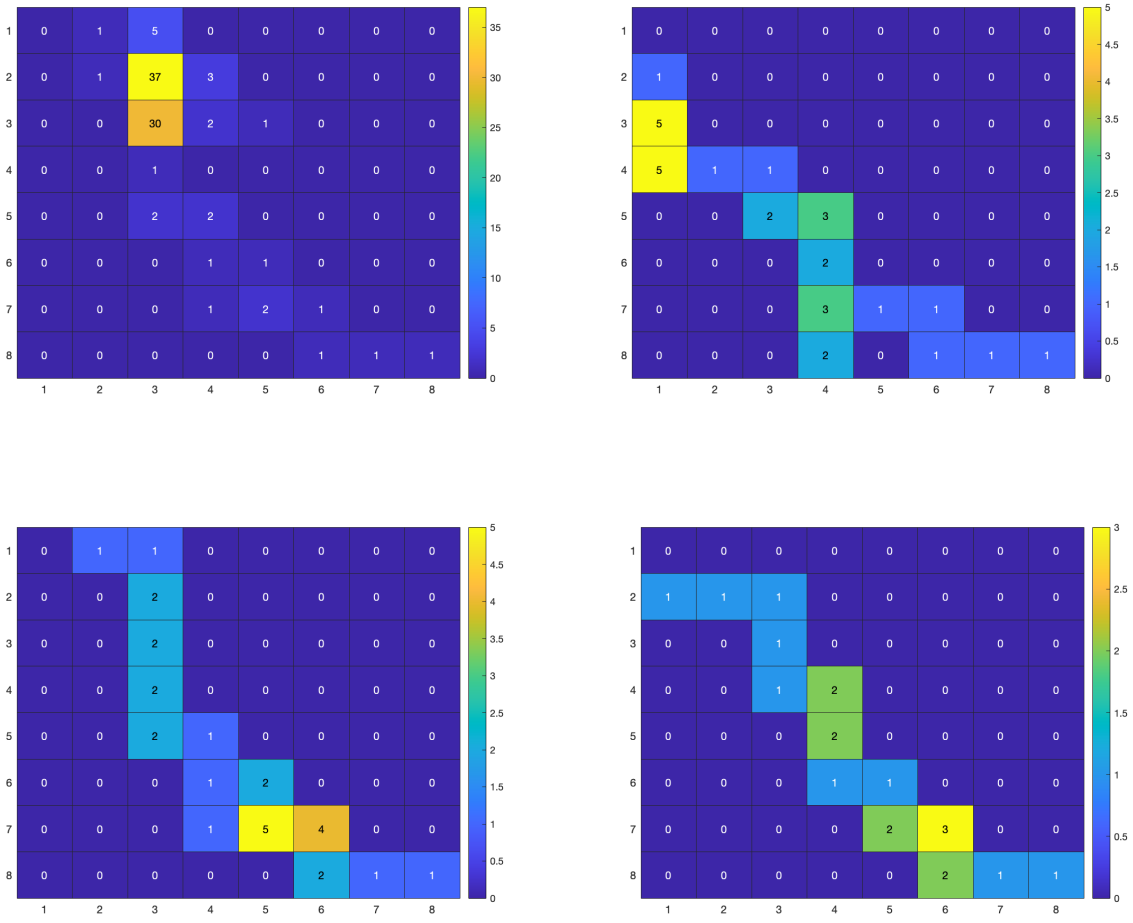


Figure 3: Heatmaps generated for (a)250 iterations, (b)1000 iterations, (c)2500 iterations, (a)4750 iterations

In the above figures, we see the heatmaps for various iterations. For early iterations, the robot takes a lot of moves and is stuck for some time in two states. In further iterations, it learns and takes less moves in a more optimal way. In the 3rd and 4th heatmaps, we see very optimal paths, and less moves, thus showing that the robot is successfully learning.

4. Extra Credit

1.Effect of different parameters:

- Keeping all parameters same, the random factor starts with 0.25, but is depreciated **every** iteration, instead of every few.

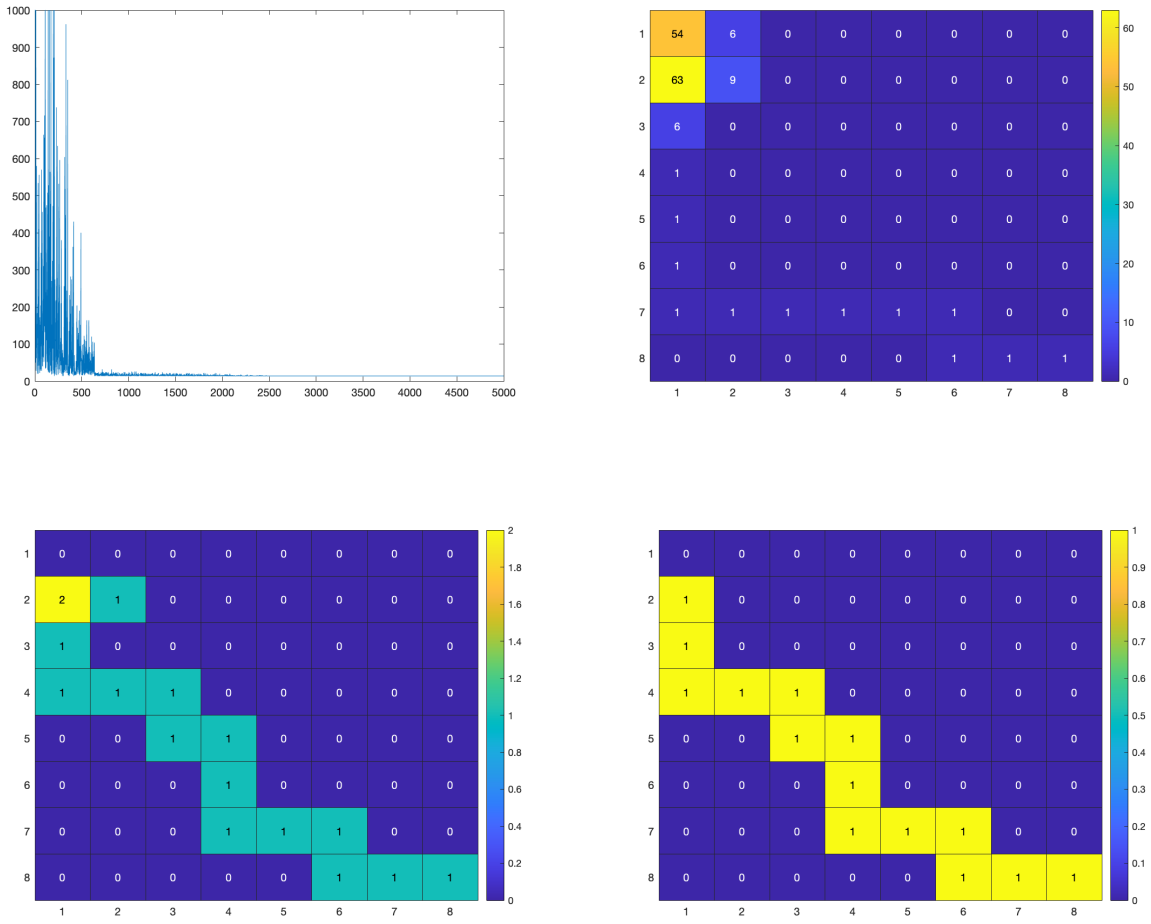


Figure 4: Number of moves, and Heatmaps generated for (a)250 iterations, (b)2000 iterations, (c)5000 iterations

The above figure shows that depreciating the random factor this rapidly forces the robot to exploit and converge very quickly. The number of moves comes down and becomes constant after a while. The optimal path is also achieved very quickly.

- Keeping all parameters same, the alpha value is increased from 0.01 to 0.1.

As can be seen from fig 5, the high alpha value changes the expectation values very rapidly, thus making the robot vary its states too much and take too many steps in between. The heatmaps also show the same. For 500th iteration, we see a lot of moves, but for the 1250th iteration, the moves go down. We think this could be optimal, but the 3500th iteration again shows increased moves and non-optimal pathways. This shows the alpha value matters a lot in the learning converging.

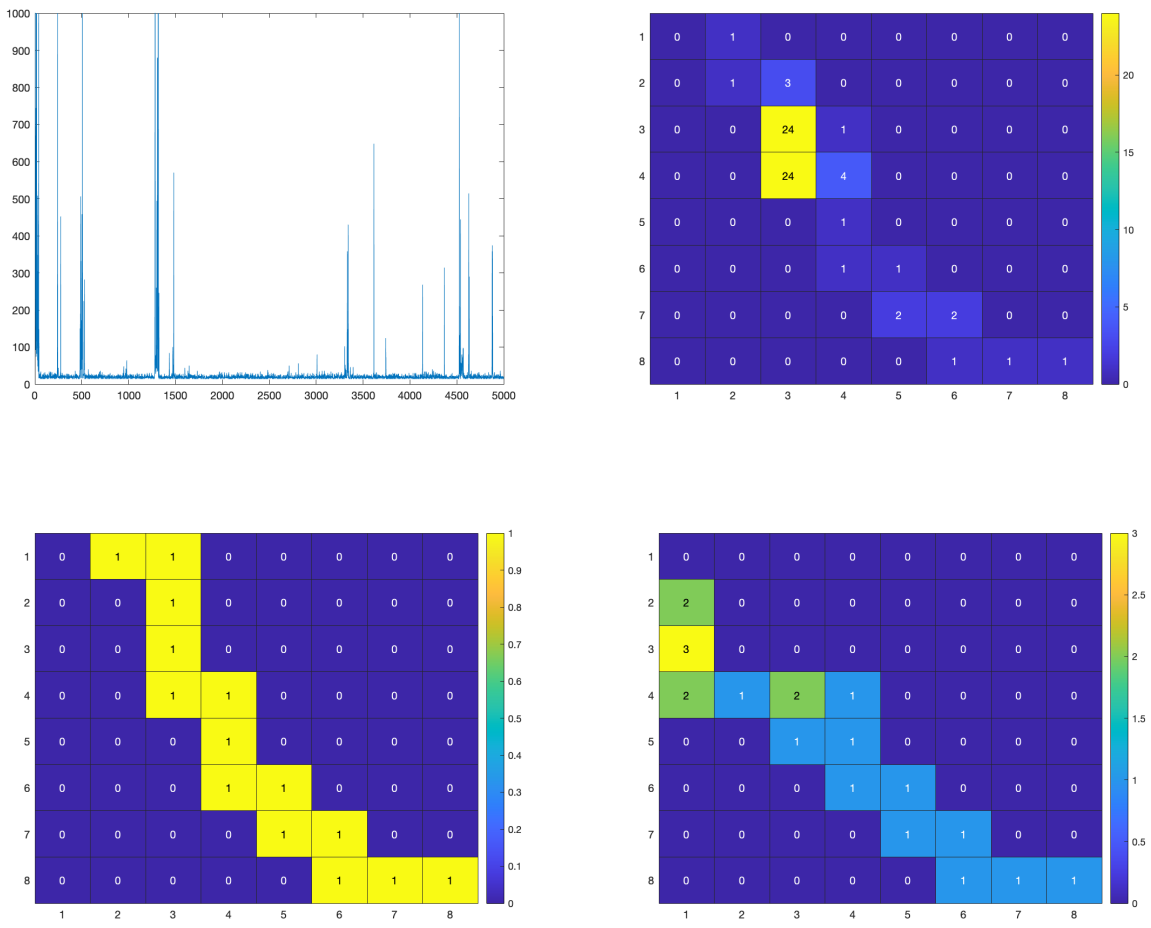
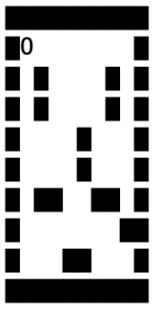


Figure 5: Number of moves, and Heatmaps generated for (a)500 iterations, (b)1250 iterations, (c)3500 iterations

2. Various maze size and patterns:

- New maze design(more walls):



```
[2, 0, 0, 0, 0, 0, 0, 0
 0, 1, 0, 0, 0, 0, 1, 0
 0, 1, 0, 0, 0, 0, 1, 0
 0, 0, 0, 0, 1, 0, 0, 0
 0, 0, 0, 0, 1, 0, 0, 0
 0, 1, 1, 0, 0, 1, 1, 0
 0, 0, 0, 0, 0, 0, 0, 1
 0, 0, 0, 1, 1, 0, 0, 0]
```

Figure 6: Maze and design with more walls

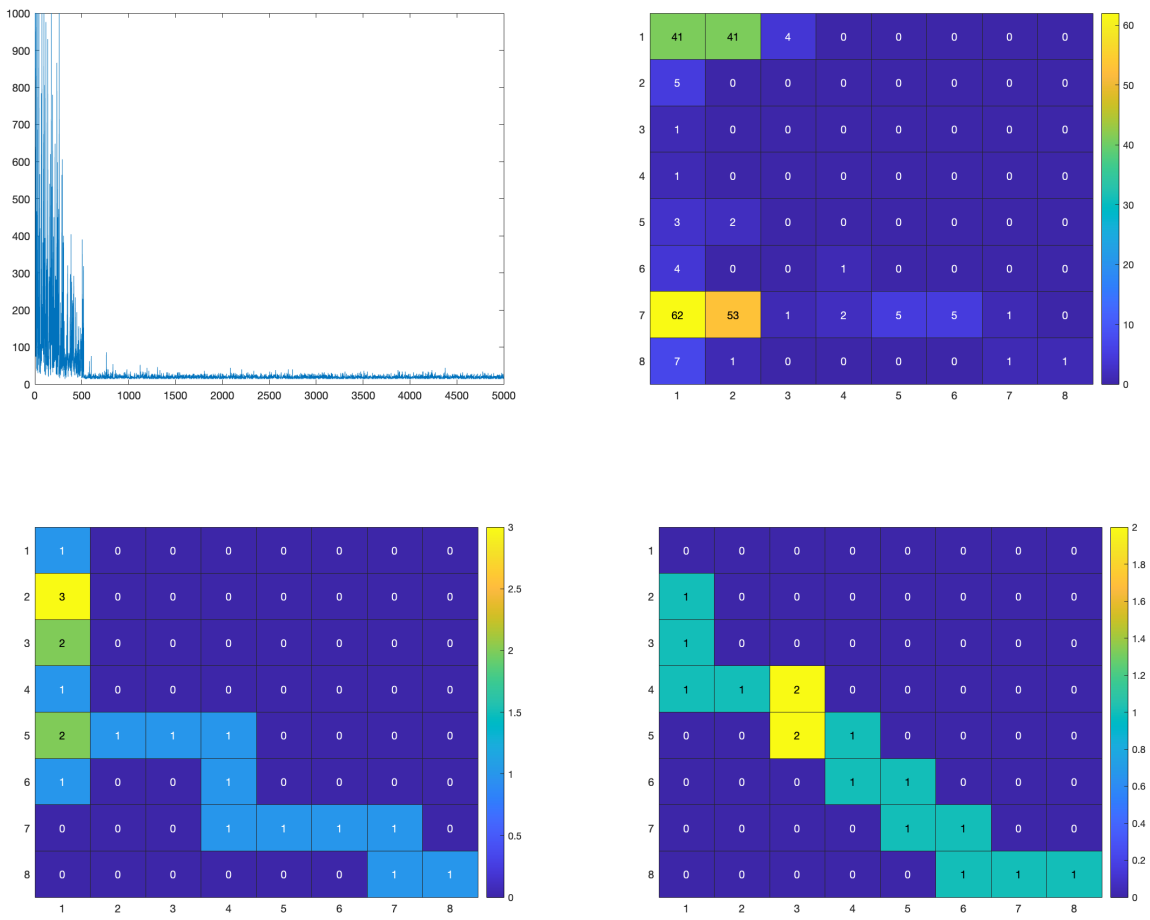


Figure 7: Number of moves, and Heatmaps generated for (a)250 iterations, (b)1750 iterations, (c)5000 iterations

Adding more walls limits the paths the robot can possibly take, thus it takes a little more time in the start to learn and find optimal solutions. The heatmap for 250th iteration shows a significantly higher number of moves(as compared to our original maze and original parameters).

- New maze design(Limited space):

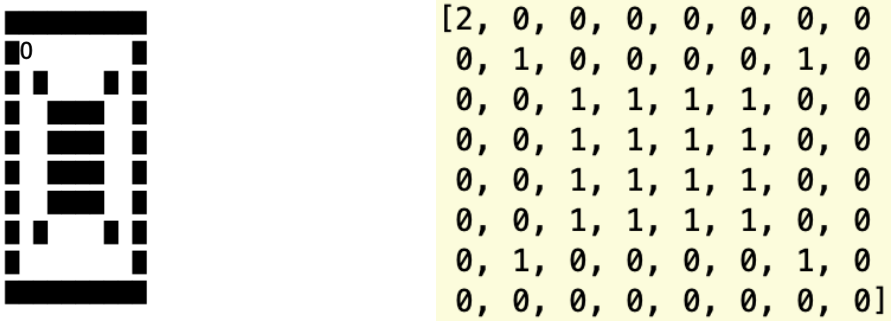


Figure 8: Maze and design with middle area blocked, limiting the space

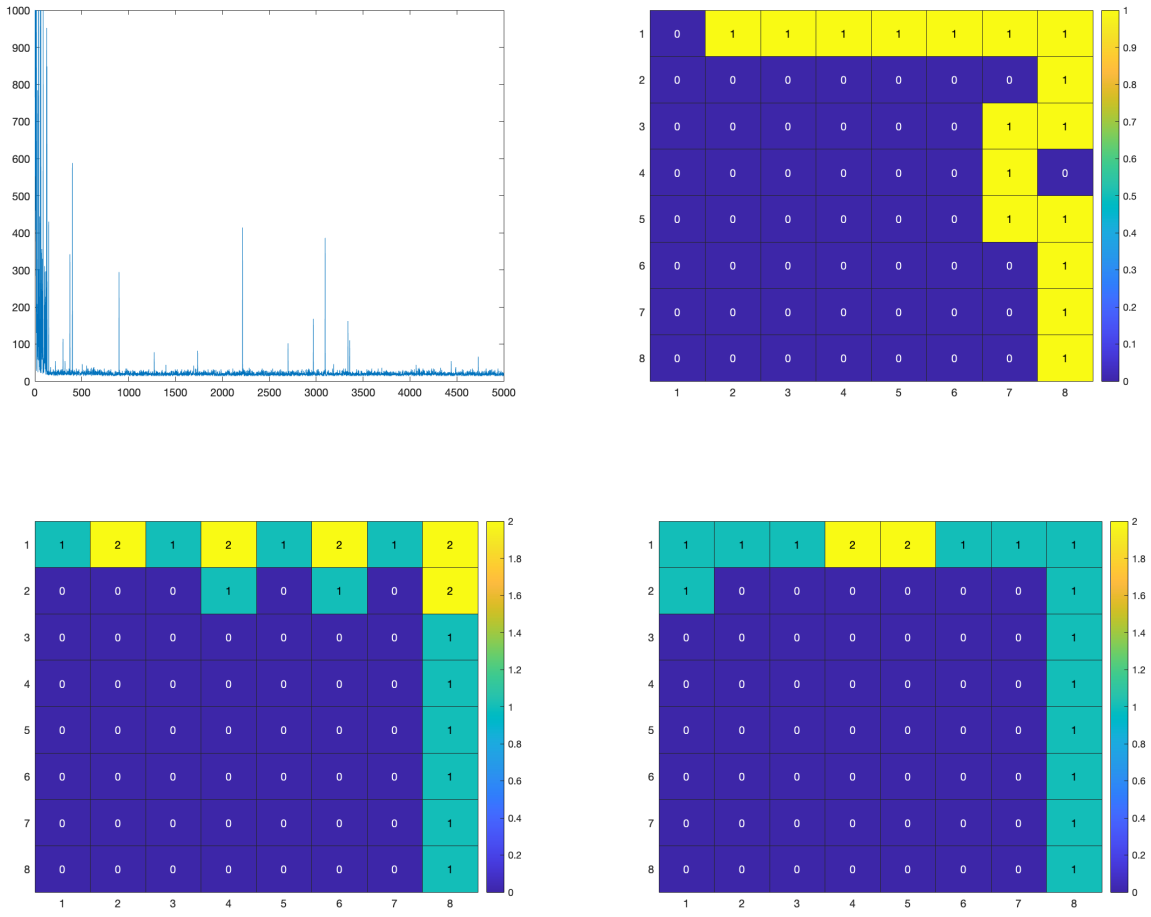
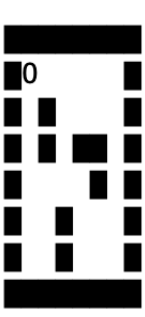


Figure 9: Number of moves, and Heatmaps generated for (a)1500 iterations, (b)4000 iterations, (c)5000 iterations

Blocking up the middle space, the robot will learn very fast, but the exploration and exploitation will not serve much of a purpose, so thats why we see peaks in number of moves and heatmaps have repeats, because there isn't much to explore to.

- New maze size(6*6):



```
[2, 0, 0, 0, 0, 0
0, 1, 0, 0, 0, 0
0, 1, 0, 1, 1, 0
0, 0, 0, 0, 1, 0
0, 0, 1, 0, 0, 0
0, 0, 1, 0, 0, 0];
```

Figure 10: Maze and design with smaller maze size

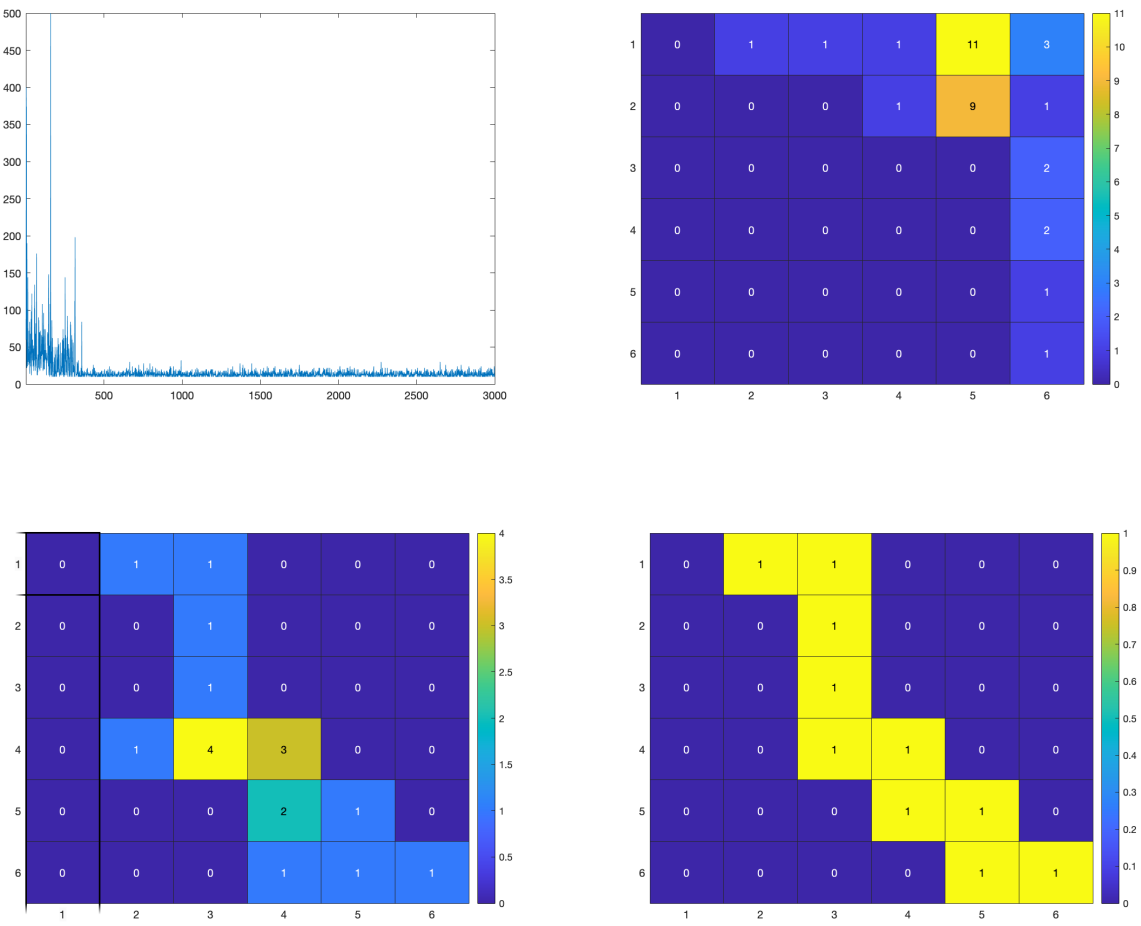


Figure 11: Number of moves, and Heatmaps generated for (a)250 iterations, (b)2000 iterations, (c)3000 iterations

We have constructed a new maze with size 6*6, and parameters as follows:
random factor = 0.25, alpha = 0.002
max steps possible = 500, number of iterations = 3000
The robot completes the maze fairly quickly given the smaller size.