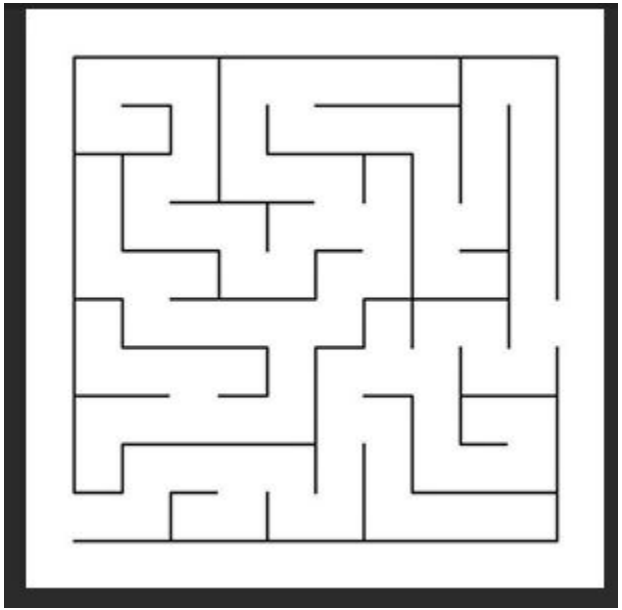
MAZE

----The Shortest Path---

Fatema Nagori 19635

Table of Content:

Introduction



In this Project we solve maze using Depth First Search(DFS) approach, we also demonstrate manual solution to find the shortest path of MAZE that has clear spaces (like Robots without wheel) using TREE and the manual solution for MAZE with unclear route(like self driving cars with wheels) using MATRIX , both the methods are used to analyse the cost, time and business it can provide in real time.

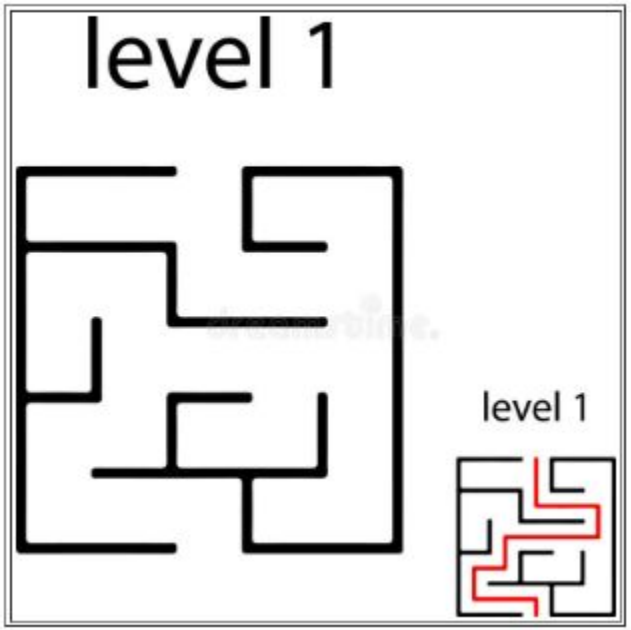
Design Approaches:

1) Depth-First Search (DFS)

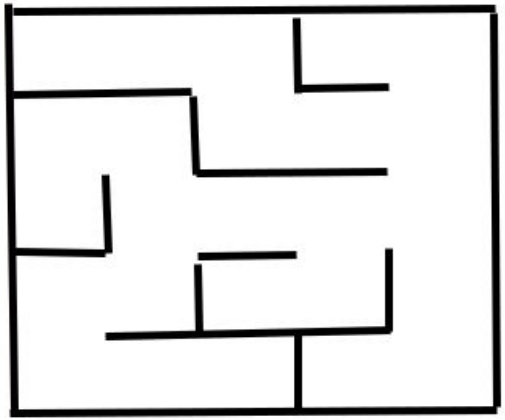
2) Breadth-First Search (BFS)

STEP 1.1 Tree-Depth First Search

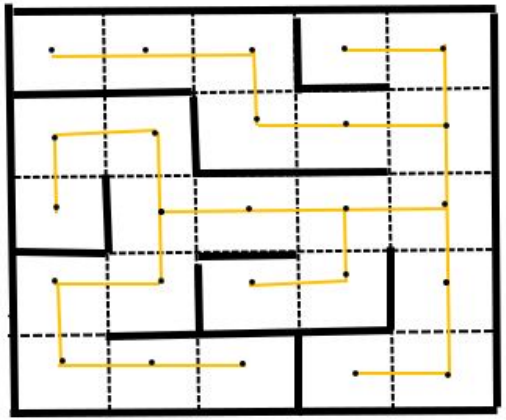
35. Conduct Depth First Traversal (DFT) on a maze - Level 1 Maze



STEP 1

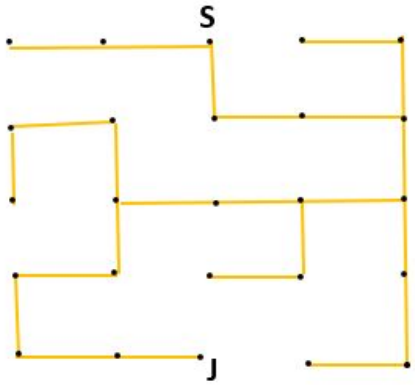


STEP 2

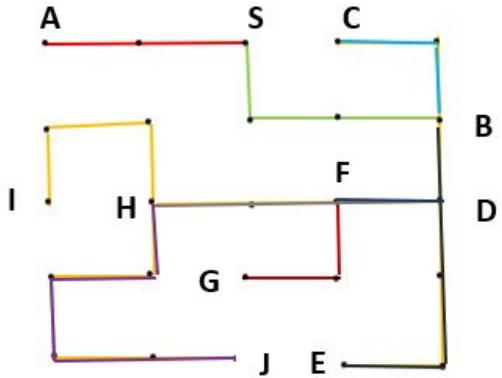


STEP 1.1
IMPLEMENTATION

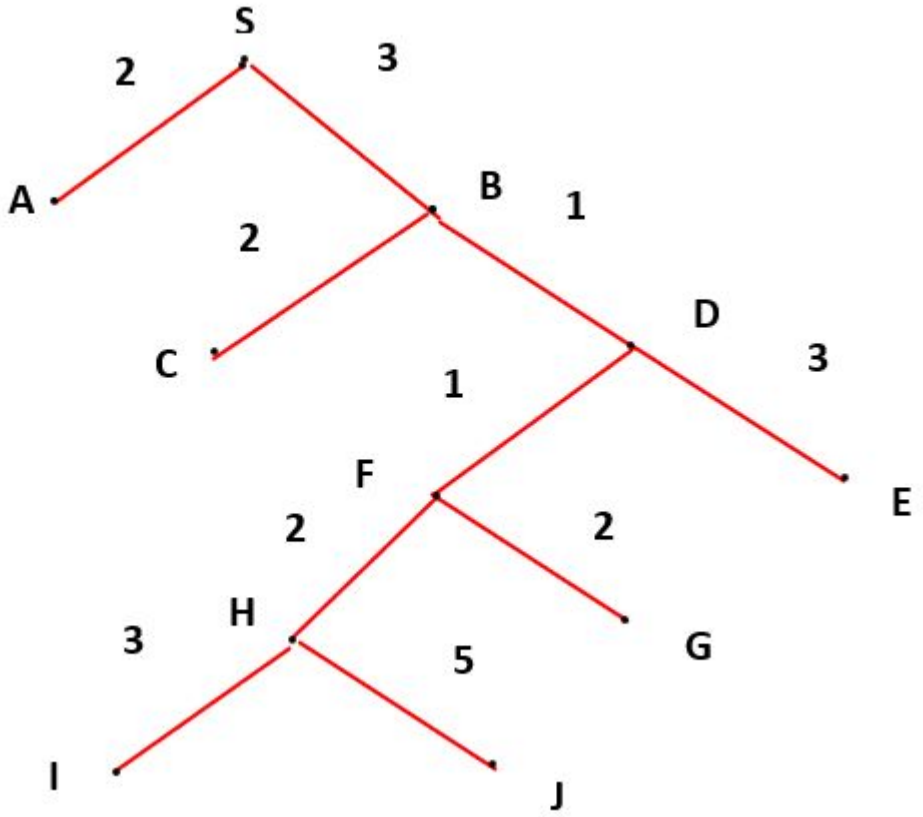
STEP 3



STEP 4

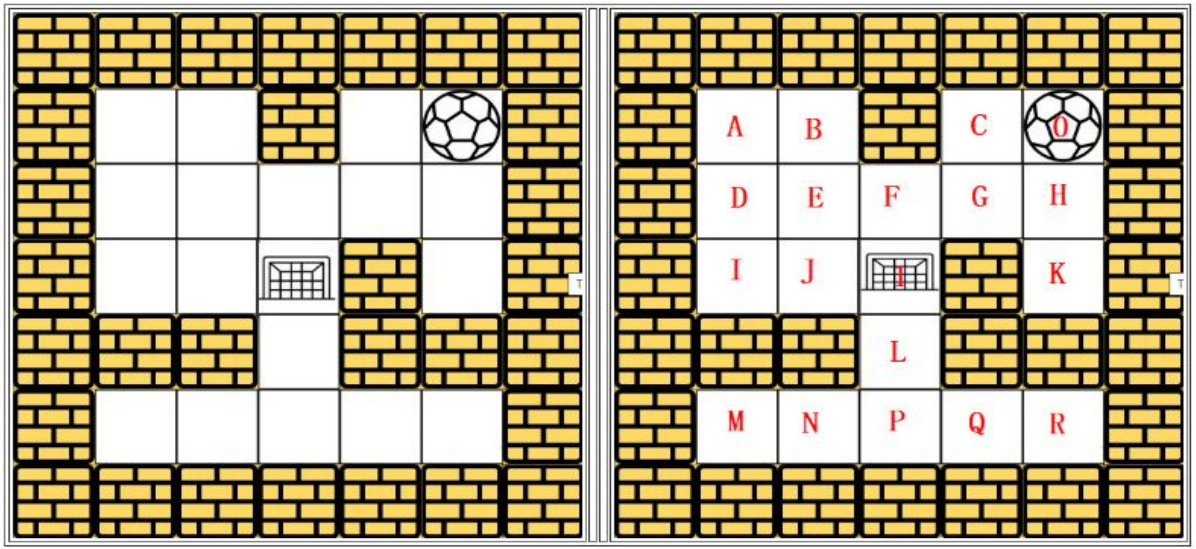


STEP 1.1 Binary Tree Depth First Search



STEP 1.2 MATRIX-Depth First Search

39. Depth-First Traversal for matrix maze
    - Please refer the concepts shown on Maze to draw the detailed steps on using Depth-First Traversal to find the pa

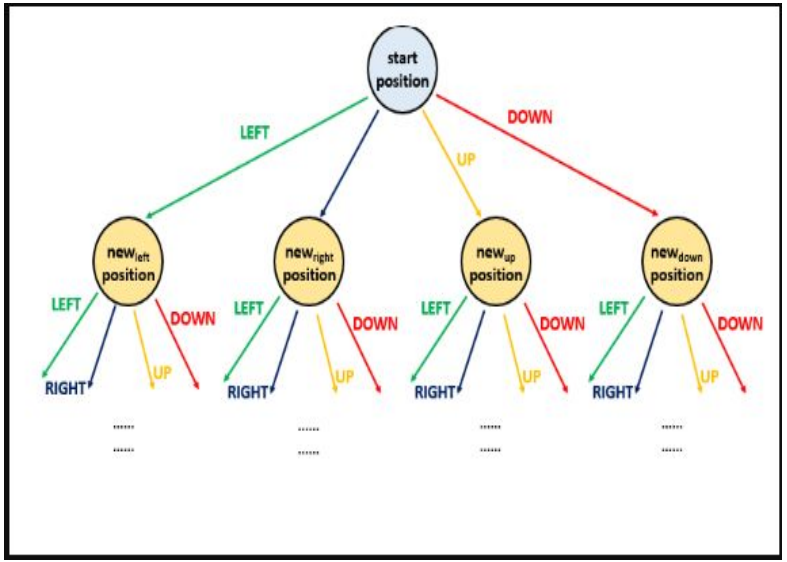


- The search sequence is

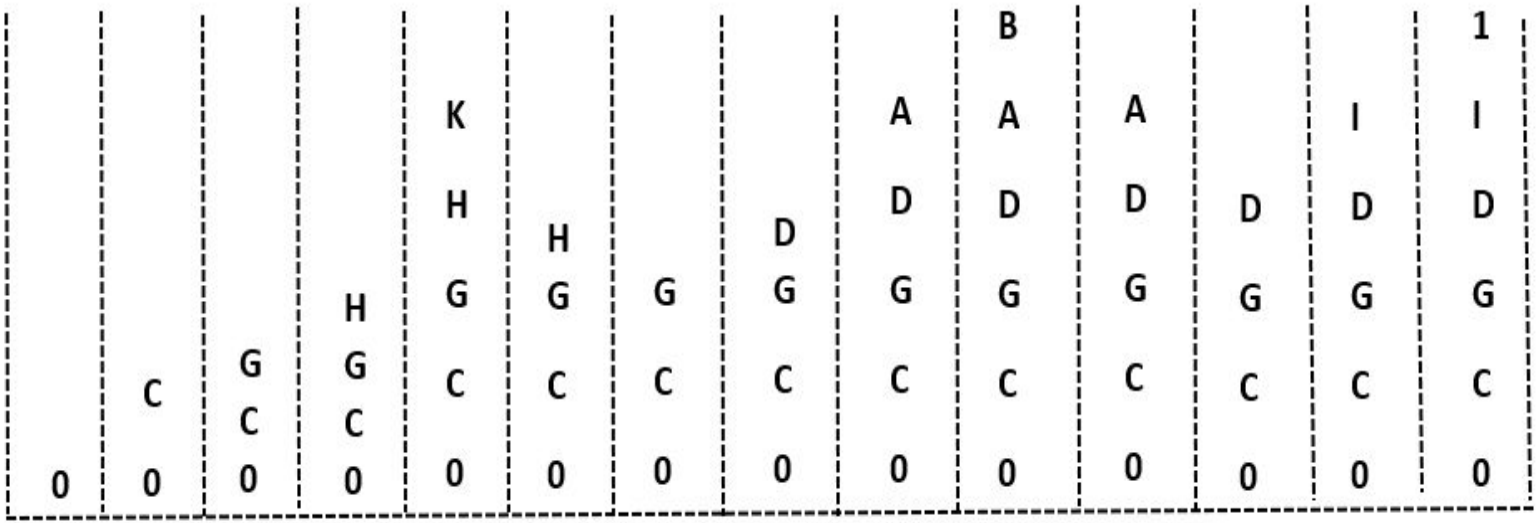
```
Right ==> Left ==> Top ==> Bottom
```

We can view the given search space in the form of a tree.

- The root node of the tree represents the starting position.
- Four different routes are possible from each position i.e. right, left, up or down.
- These four options can be represented by 4 branches of each node in the given tree.
- Thus, the new node reached from the root traversing over the branch represents the new position occupied by the ball after choosing the corresponding direction of travel.

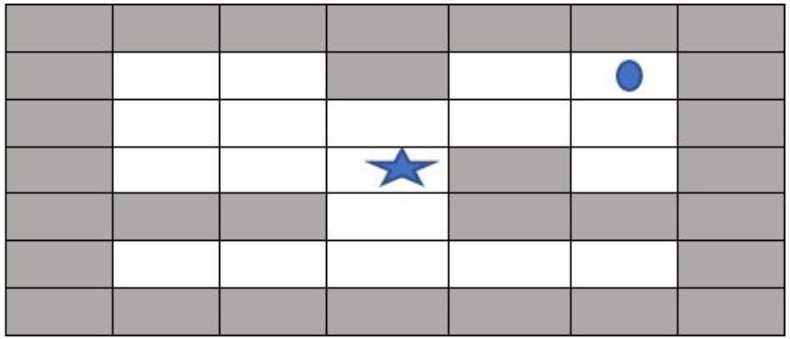


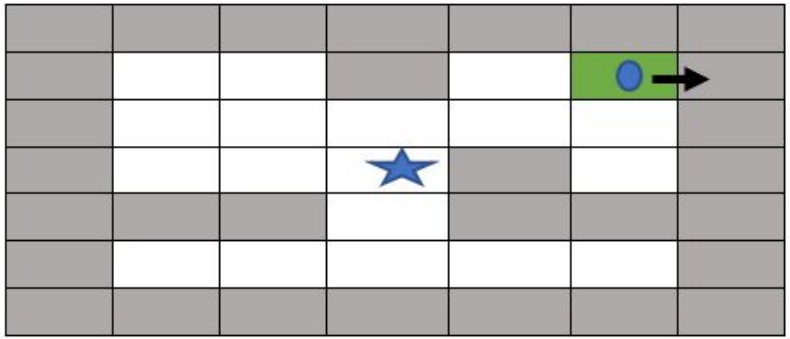
Matrix Implementation- stacks

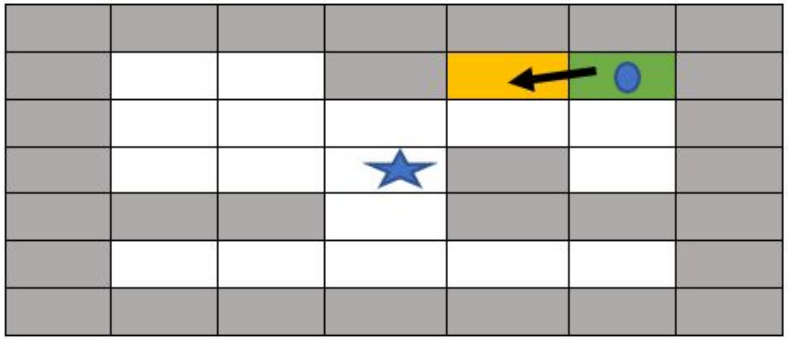


STEPS FOR 1.2 MATRIX C (Right, left, up, down)

1. Start from 0
2. 0→C
3. 0→C→G
4. 0→C→G→H
5. 0→C→G →H→K
6. 0→C→G →H (POP K, from k it cannot go anywhere)
7. 0→C→G (POP H, from H it cannot go anywhere)
8. 0→C→G →D
9. 0→C→G →D→A
10. 0→C→G →D→A→B
11. 0→C→G →D→A (POP B, from B it cannot go anywhere)
12. 0→C→G →D (POP A, from A it cannot go anywhere)
13. 0→C→G →D→I
14. 0→C→G →D→I→1 (FINAL DESTINATION, ENDS AT 1)







Wall

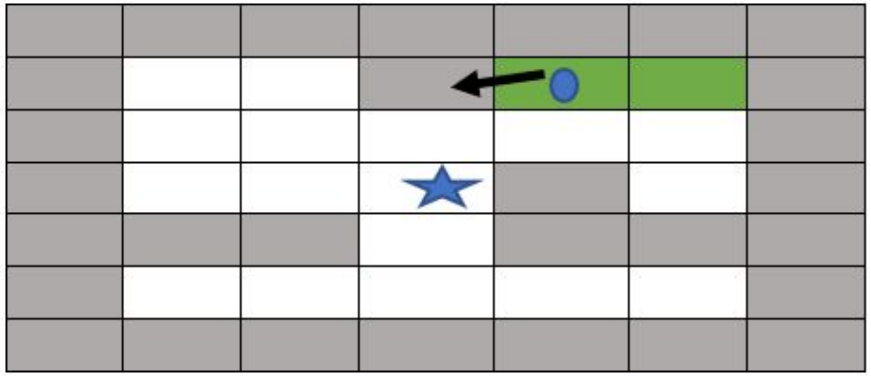
Empty Space

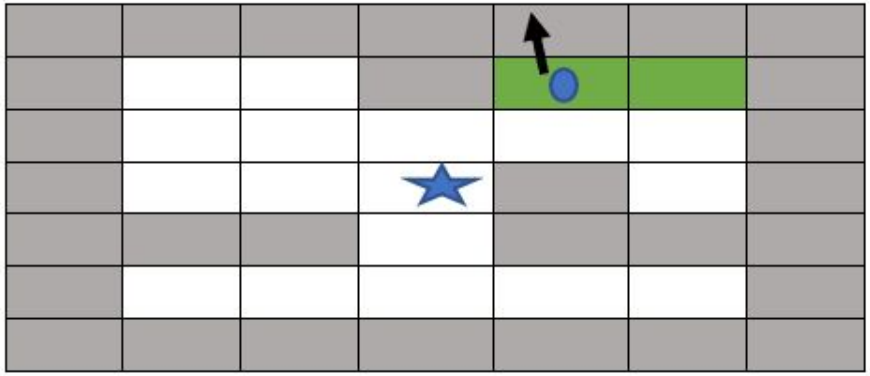
Destination

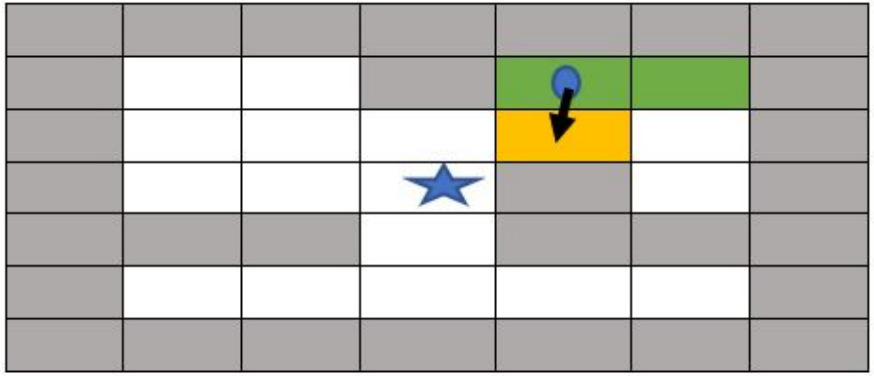
Current Position

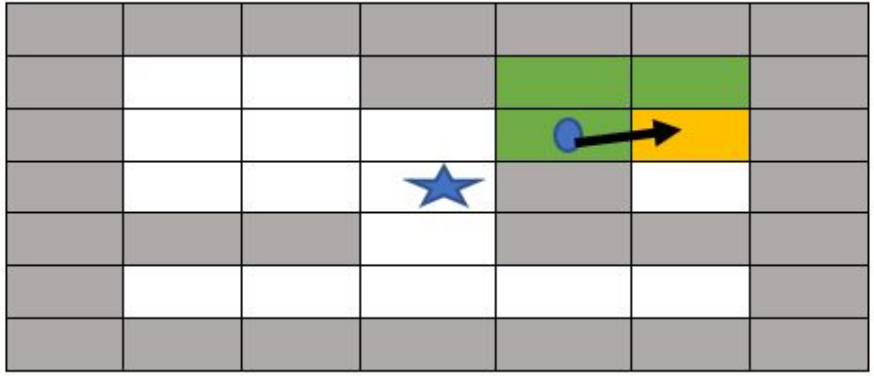
Visited

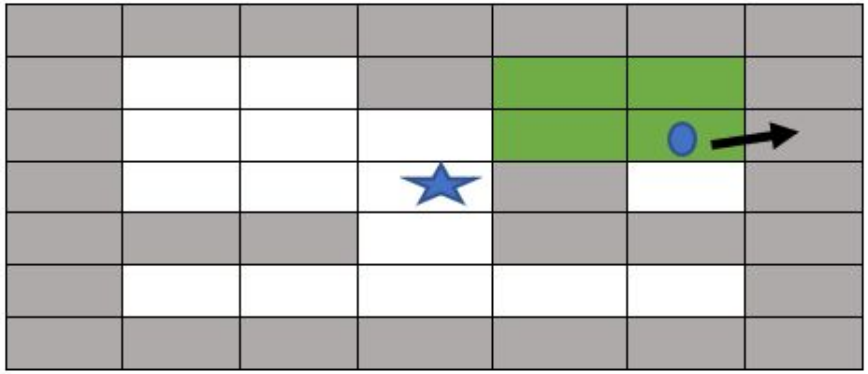
Path Traversed currently

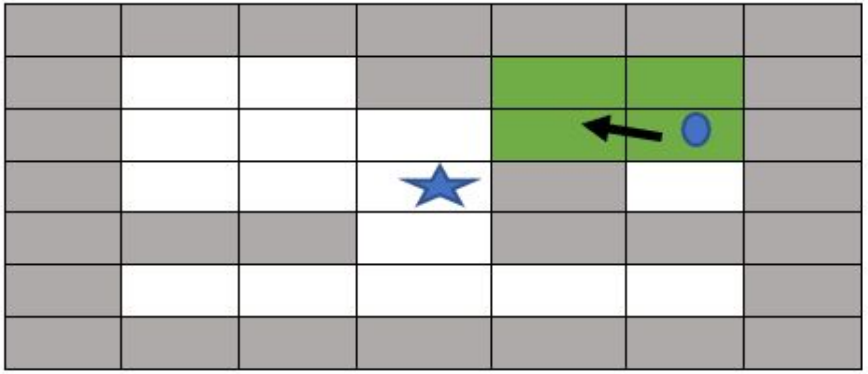


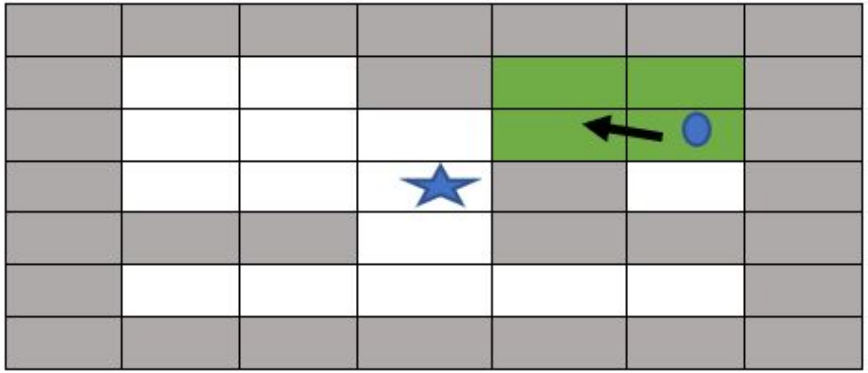


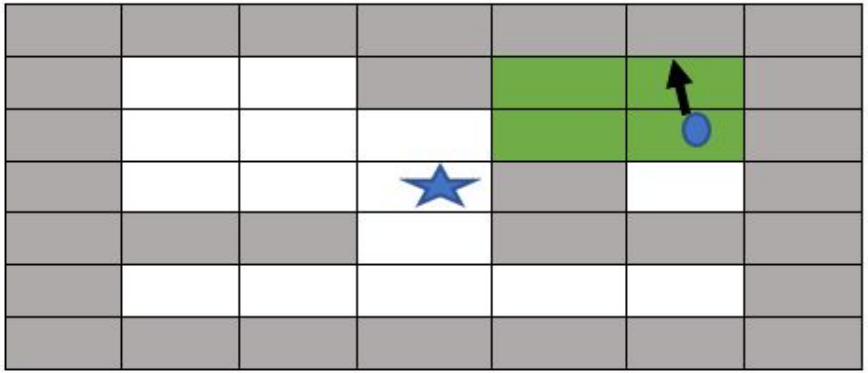


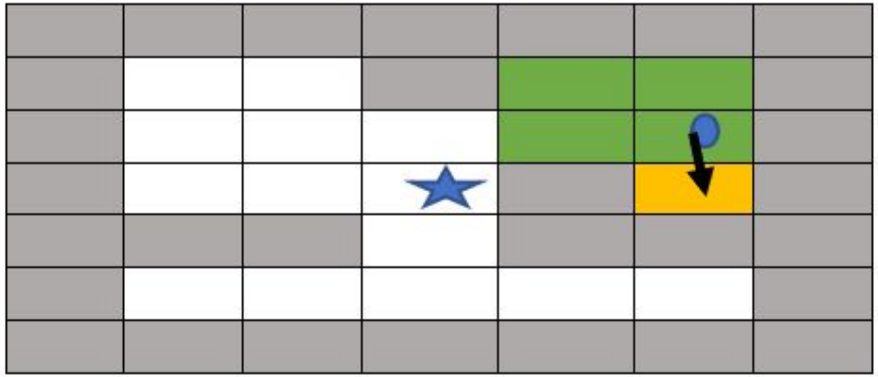


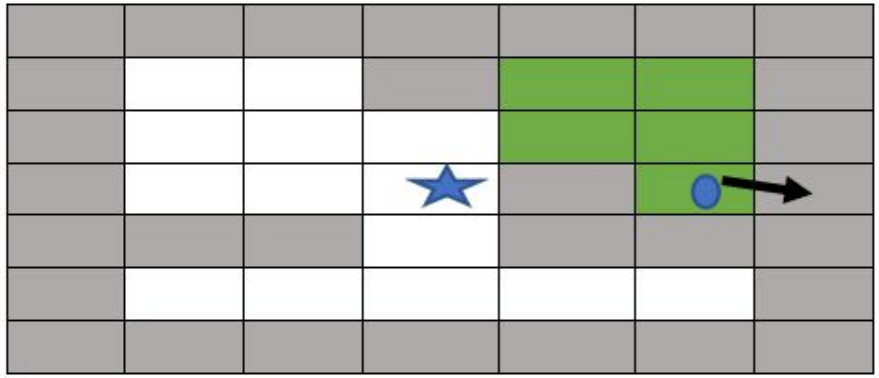


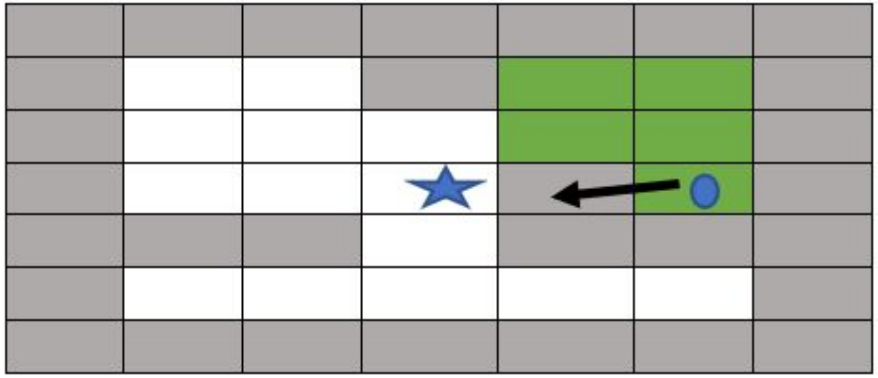


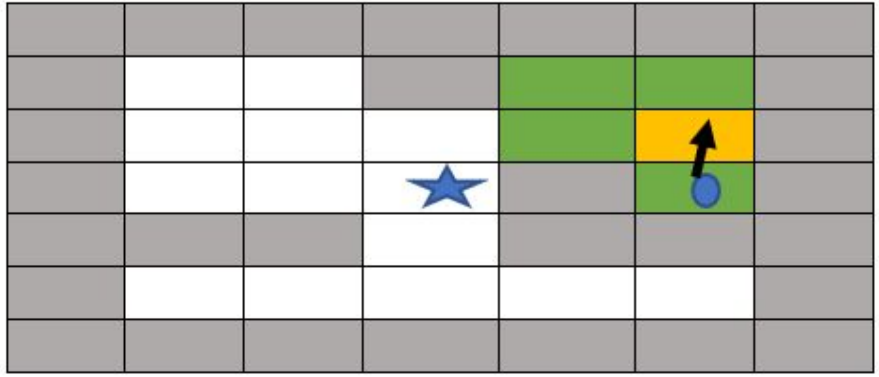


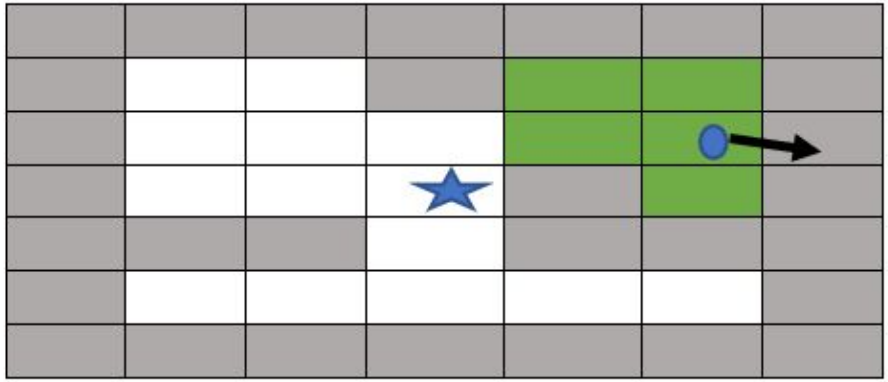


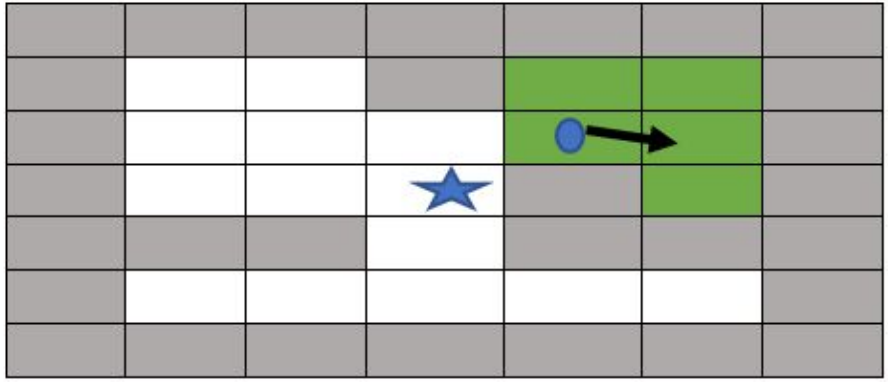


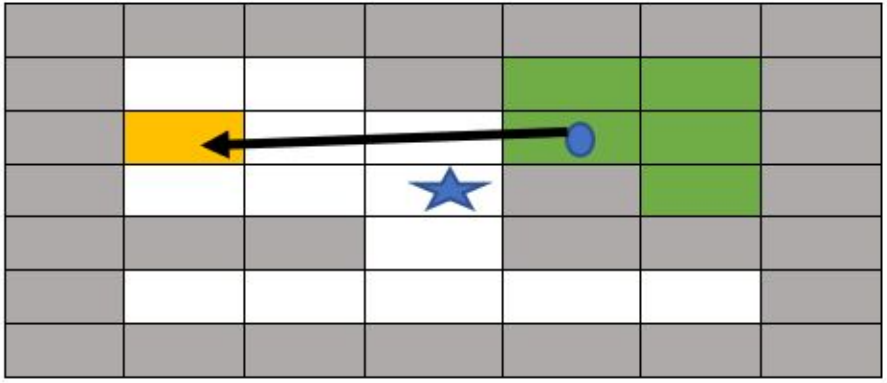


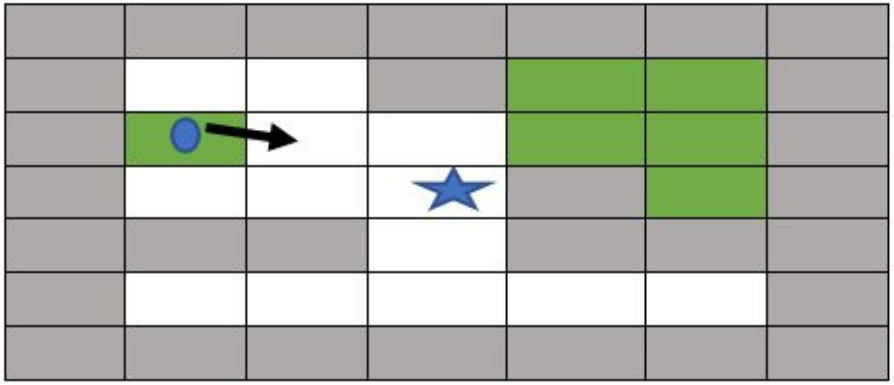


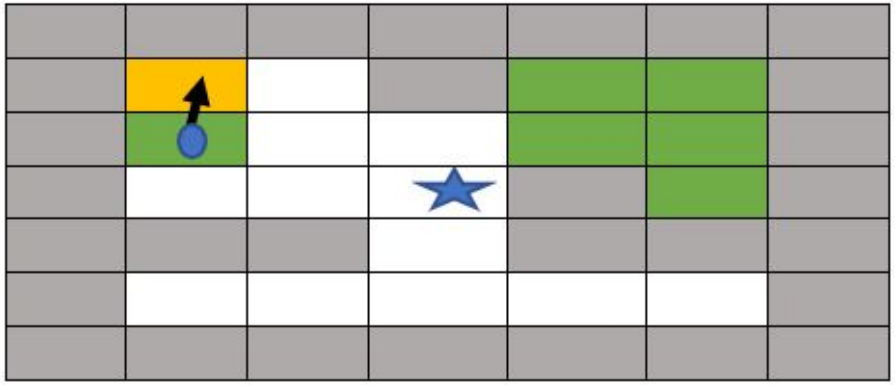


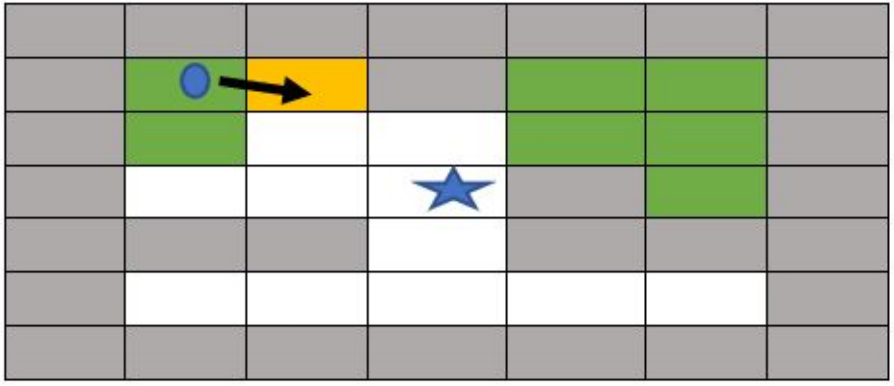


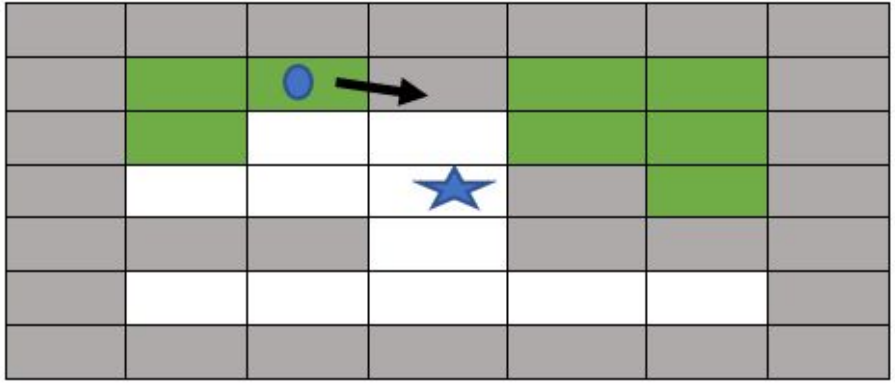


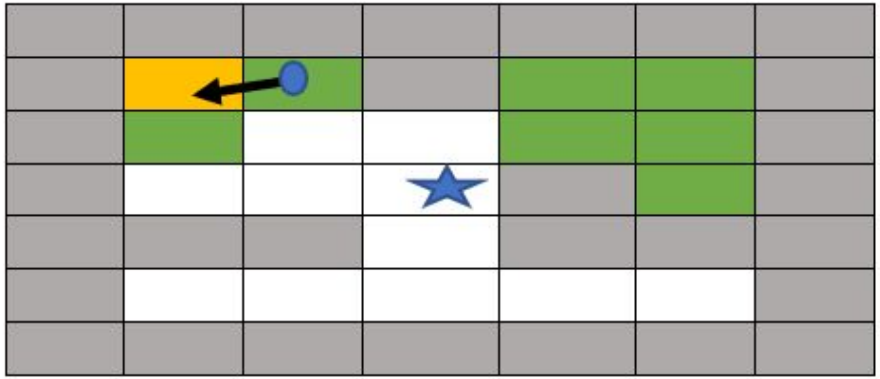


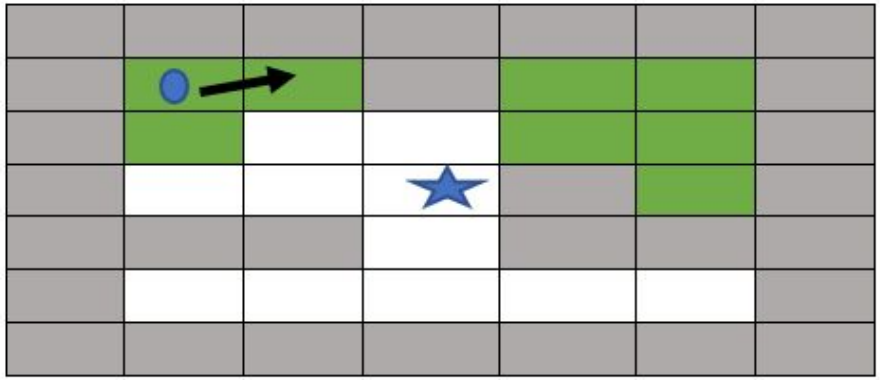


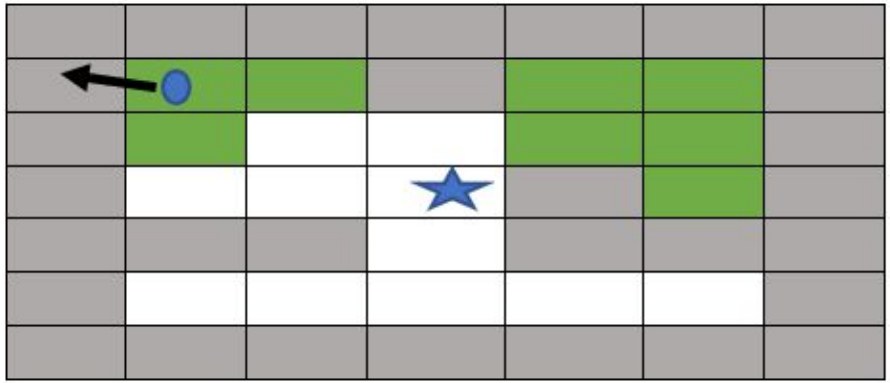


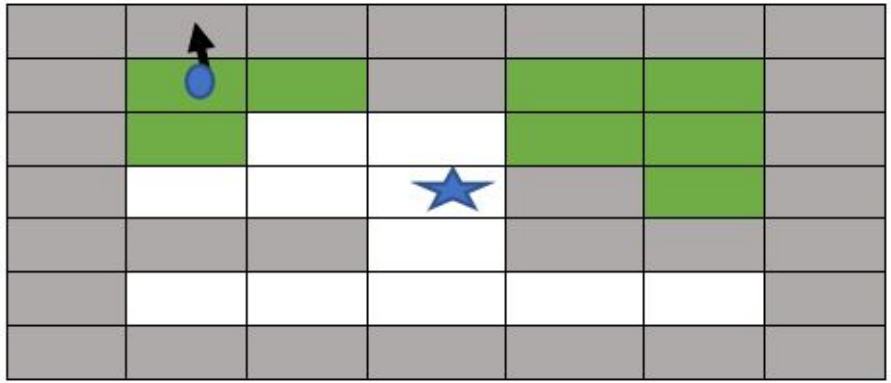


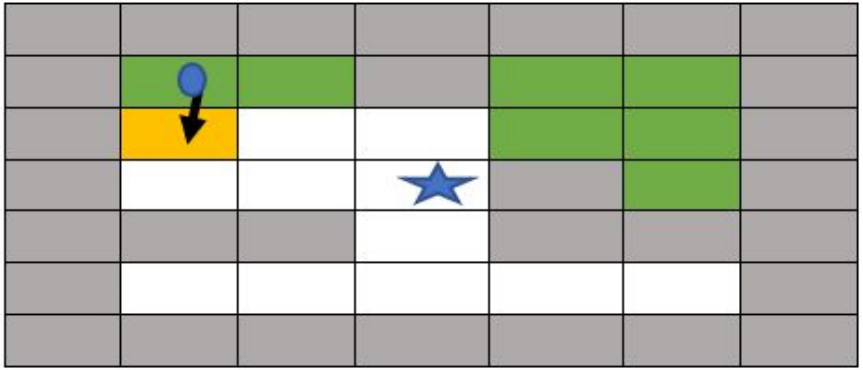


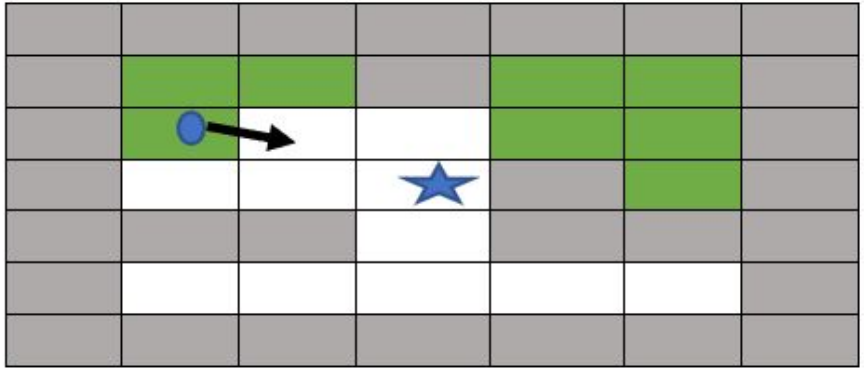


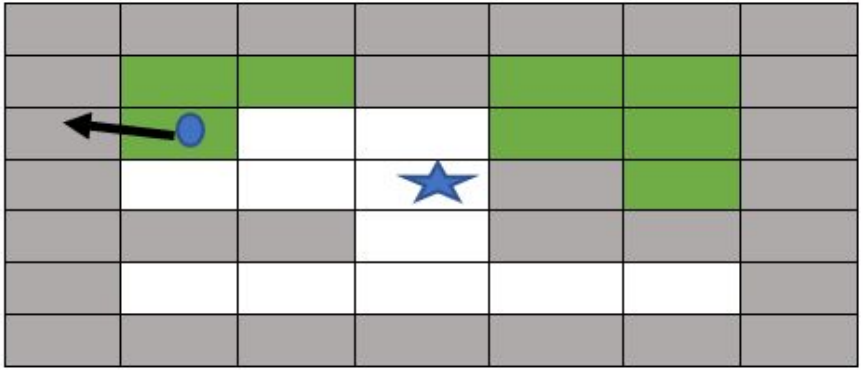


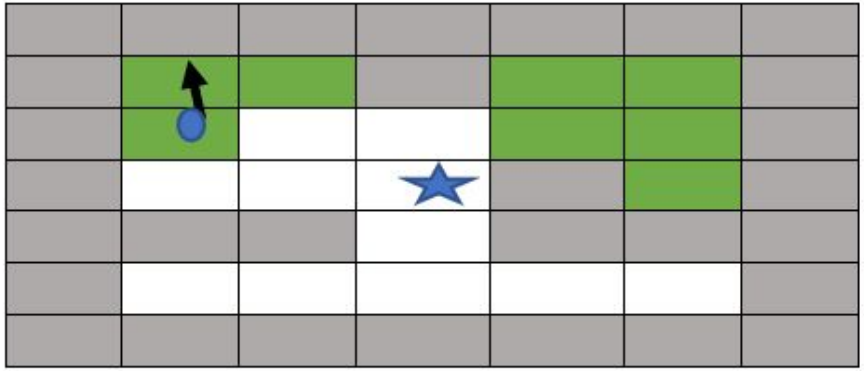


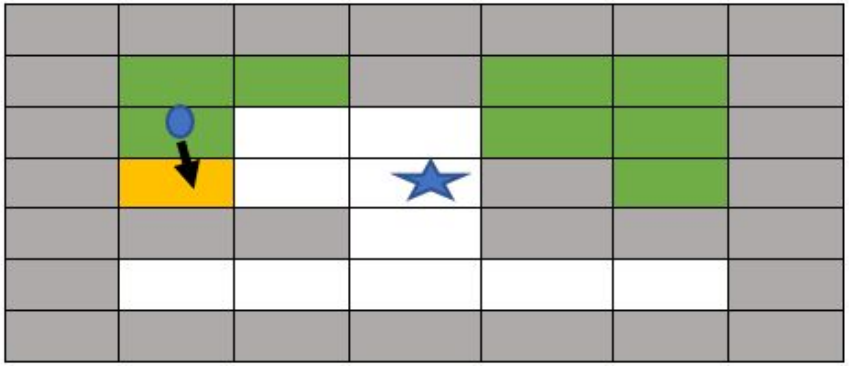


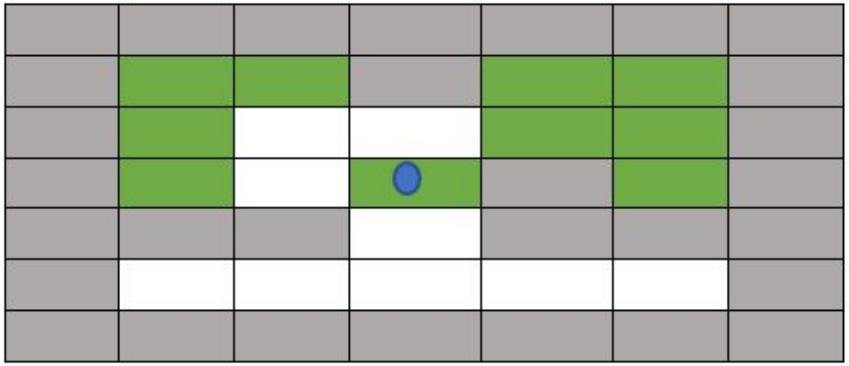






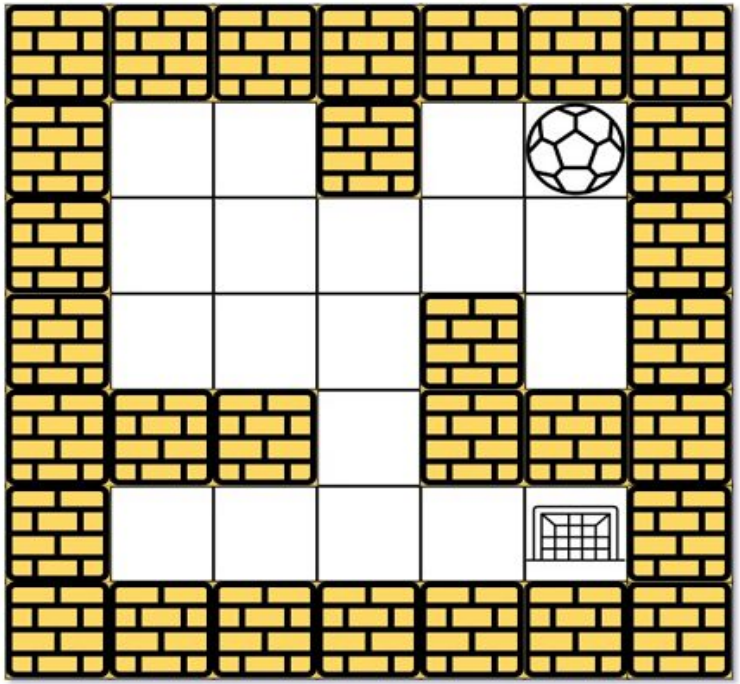






Reached final destination

STEP:2 Question 490 Leetcode: Maze

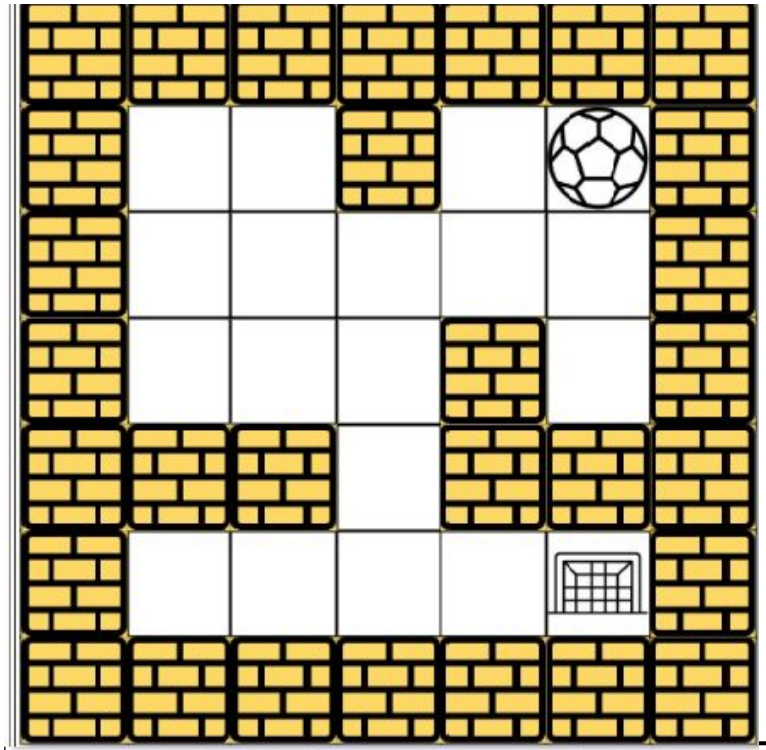


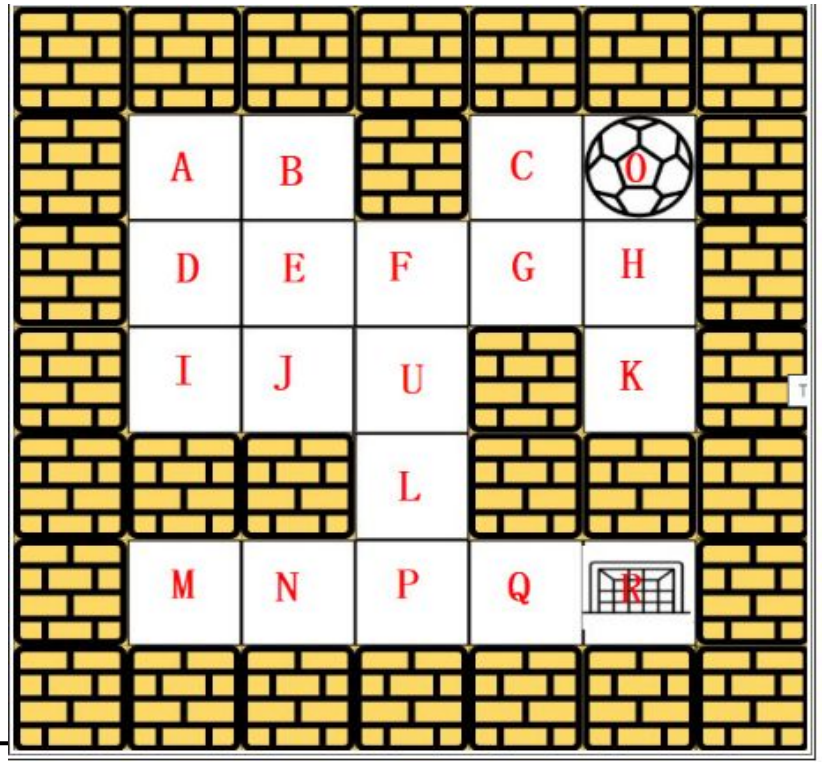
- **Given**: Empty spaces represented as 0 and wall as 1,ball rolls in direction **up, down, left, right** and stops when hits the wall and choose next direction.
- m*n maze
- Start position=[start row, start col]
- destination=[dest row, dest col]
- Return True=ball stops else false

constrain

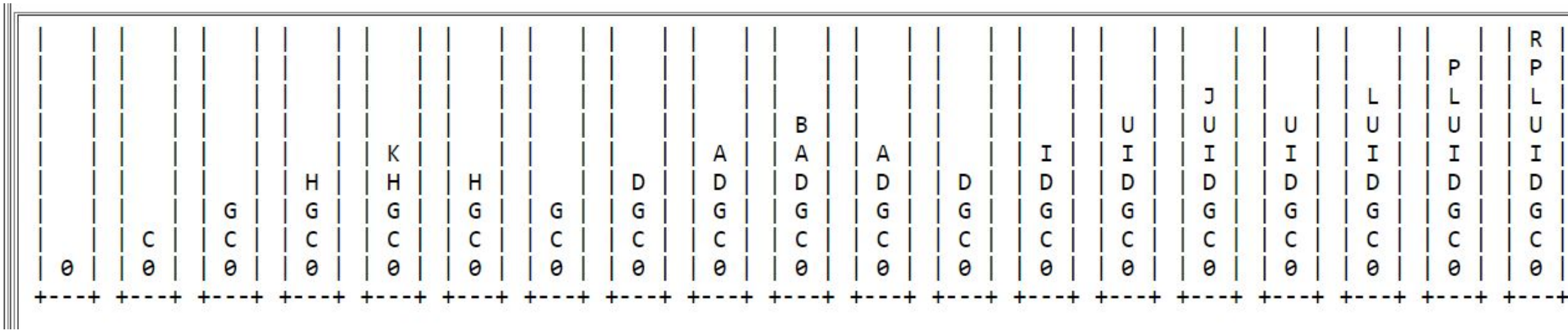
- m=maze.length
- n==maze[i].length
- 1<=m, n<=100
- Maze[i[j is 0 or 1
- start.length==2
- destination.length==2
- 0<=start row, dest row<=m
- 0<=start col, dest col<=n
- Both ball and destination exist in empty spaces and they will not be in the same position initially
- The maze contains at least 2 empty spaces

SOLUTION using DFT approach





Stacks (DFS)


0
C0
GC0
HGC0
KHGC0
HGC0
GC0
DGC0
ADGC0
BADGC0
ADGC0
DGC0
IDGC0
UIDGC0
JUIDGC0
UIDGC0
LUIDGC0
PLUIDGC0
RPLUIDGC0

Enhancement Ideas- Compare Algorithm

	TIME COMPLEXITY	SPACE
DEPTH FIRST SEARCH	O(MN)	O(MN)
BREADTH FIRST SEARCH	O(MN)	O(MN)

Test case Diagram

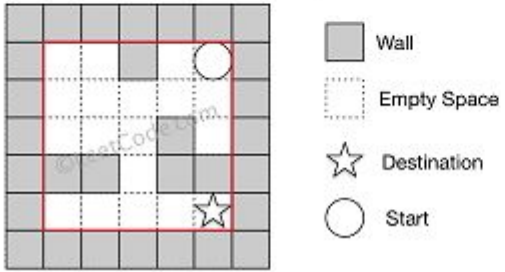
Input 1: a maze represented by a 2D array

```
0 0 1 0 0
0 0 0 0 0
0 0 0 1 0
1 1 0 1 1
0 0 0 0 0
```

Input 2: start coordinate (rowStart, colStart) = (0, 4)
Input 3: destination coordinate (rowDest, colDest) = (4, 4)

Output: true

Explanation: One possible way is : left -> down -> left -> down -> right -> down -> right.



Test case Diagram

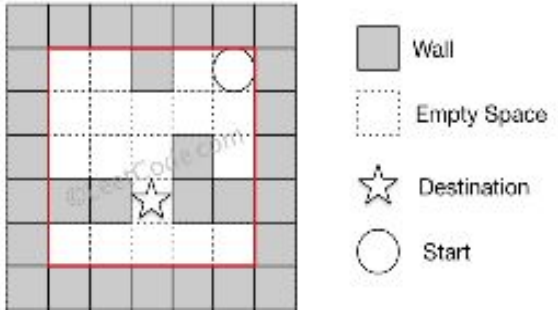
Input 1: a maze represented by a 2D array

```
0 0 1 0 0
0 0 0 0 0
0 0 0 1 0
1 1 0 1 1
0 0 0 0 0
```

Input 2: start coordinate (rowStart, colStart) = (0, 4)
Input 3: destination coordinate (rowDest, colDest) = (3, 2)

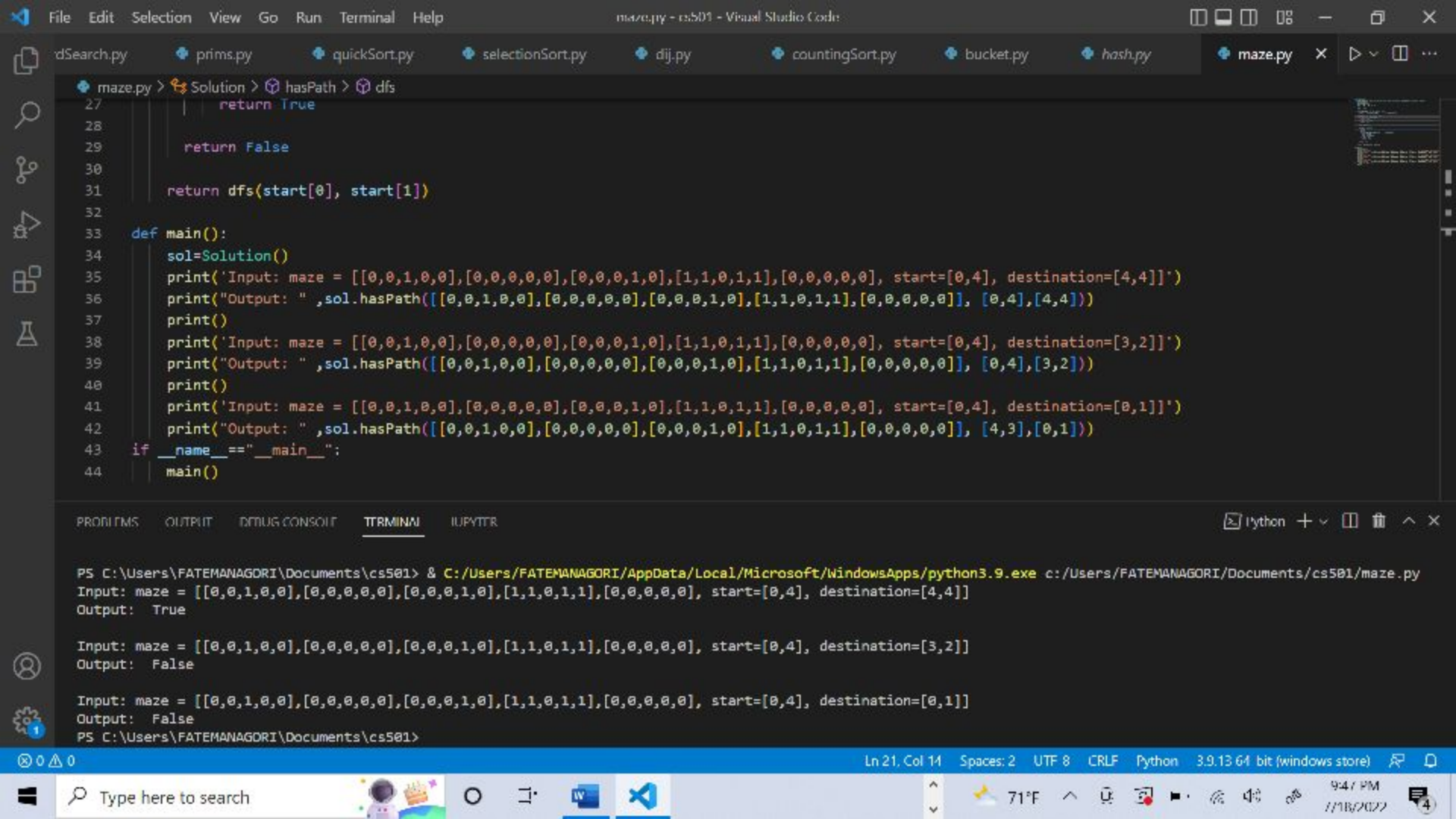
Output: false

Explanation: There is no way for the ball to stop at the destination.



maze.py > ...

```python
class Solution:
    def hasPath(self, maze: list[list[int]], start: list[int], destination: list[int]) -> bool:
        m = len(maze)
        n = len(maze[0])
        dirs = [0, 1, 0, -1, 0]

        seen = set()

        def isValid(x: int, y: int) -> bool:
            return 0 <= x < m and 0 <= y < n and maze[x][y] == 0

        def dfs(i: int, j: int) -> bool:
            if [i, j] == destination:
                return True
            if (i, j) in seen:
                return False

            seen.add((i, j))

            for k in range(4):
                x = i
                y = j
                while isValid(x + dirs[k], y + dirs[k + 1]):
                    x += dirs[k]
                    y += dirs[k + 1]
                if dfs(x, y):
                    return True

            return False

```


File Edit Selection View Go Run Terminal Help
maze.py - cs501 - Visual Studio Code
dSearch.py
prims.py
quickSort.py
selectionSort.py
dij.py
countingSort.py
bucket.py
hash.py
maze.py
maze.py > Solution > hasPath > dfs
return True
return False
return dfs(start[0], start[1])
def main():
sol=Solution()
print('Input: maze = [[0,0,1,0,0],[0,0,0,0,0],[0,0,0,1,0],[1,1,0,1,1],[0,0,0,0,0], start=[0,4], destination=[4,4]]')
print("Output: " ,sol.hasPath([[0,0,1,0,0],[0,0,0,0,0],[0,0,0,1,0],[1,1,0,1,1],[0,0,0,0,0]], [0,4],[4,4]))
print()
print('Input: maze = [[0,0,1,0,0],[0,0,0,0,0],[0,0,0,1,0],[1,1,0,1,1],[0,0,0,0,0], start=[0,4], destination=[3,2]]')
print("Output: " ,sol.hasPath([[0,0,1,0,0],[0,0,0,0,0],[0,0,0,1,0],[1,1,0,1,1],[0,0,0,0,0]], [0,4],[3,2]))
print()
print('Input: maze = [[0,0,1,0,0],[0,0,0,0,0],[0,0,0,1,0],[1,1,0,1,1],[0,0,0,0,0], start=[0,4], destination=[0,1]]')
print("Output: " ,sol.hasPath([[0,0,1,0,0],[0,0,0,0,0],[0,0,0,1,0],[1,1,0,1,1],[0,0,0,0,0]], [4,3],[0,1]))
if __name__=="__main__":
main()
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
JUPYTER
Python
PS C:\Users\FATEMANAGORI\Documents\cs501> & C:/Users/FATEMANAGORI/AppData/Local/Microsoft/WindowsApps/python3.9.exe c:/Users/FATEMANAGORI/Documents/cs501/maze.py
Input: maze = [[0,0,1,0,0],[0,0,0,0,0],[0,0,0,1,0],[1,1,0,1,1],[0,0,0,0,0], start=[0,4], destination=[4,4]]
Output: True
Input: maze = [[0,0,1,0,0],[0,0,0,0,0],[0,0,0,1,0],[1,1,0,1,1],[0,0,0,0,0], start=[0,4], destination=[3,2]]
Output: False
Input: maze = [[0,0,1,0,0],[0,0,0,0,0],[0,0,0,1,0],[1,1,0,1,1],[0,0,0,0,0], start=[0,4], destination=[0,1]]
Output: False
PS C:\Users\FATEMANAGORI\Documents\cs501>
Ln 21, Col 14
Spaces: 2
UTF-8
CRLF
Python
3.9.13 64-bit (windows store)
Type here to search
71°F

Conclusion

The shortest path can be find using Depth-First Search and Breadth-First Search . However In this project Depth-First Search is used to find shortest path successfully. For the Leetcode question both the approaches takes same time and space, but in general For mazes specifically (if we define a maze as there being only one way to reach a cell from the starting point without backtracking, meaning it's essentially a tree), BFS will generally use more memory, as we'll need to keep multiple paths in memory at the same time, where DFS only needs to keep track of a single path at any given time.

Reference

- Prof Chang's Class material CS501
- Tree/Matrix
- LeetCode