CSINTSY MCO1: MazeBot

Members:

Asturiano, Christian Emmanuel S.
Cheng, Samuel Vincent T.
Custer, Mark John T.
De Ramos, Ghrazielle Rei A.

Submitted to:

Sir Thomas James Tiam-Lee

March 2023

I. Introduction

Given an n x n maze and the locations of the start tile and goal tile, the task is to create a maze bot that successfully finds a path from the start to the goal. Certain tiles are walls, meaning they are inaccessible to the bot.The maze bot may only move one step up, down, left, or right at a time. The maze bot uses the A* Search Algorithm to find the optimal path from start to goal, if any path exists. It uses the Manhattan distance between a tile and the goal as its heuristic function.

II. Program

Note: The program was written in Python 3.

1. Place the source code and the text file containing the maze in the same directory
2. Run the python script "MazeBot.py" with the command: "python MazeBot.py"
3. Upon running the program, the search algorithm is run, then the total number of tiles explored will be printed to the terminal and a GUI window opens to display the maze.

Display Rules:

The GUI window uses different colors to highlight different elements of the maze:

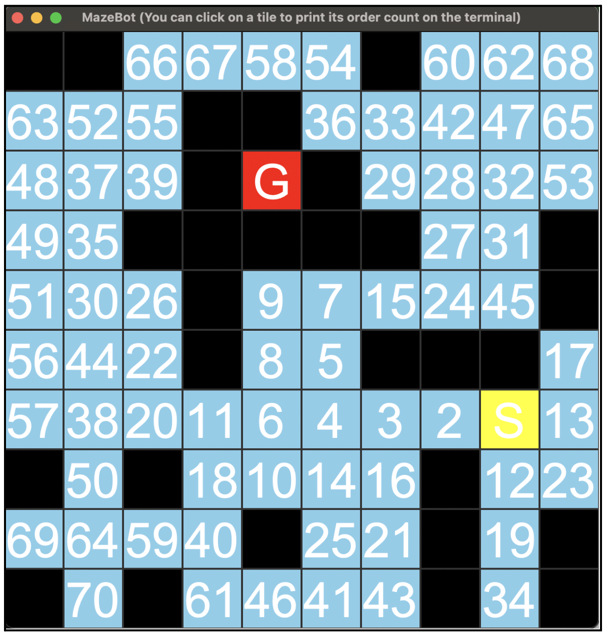
a. Every explored tile contains a number indicating the order in which it was explored.
b. Clicking on an explored tile prints the order in which it was explored to the terminal.
a. The start state node is colored yellow.
b. The goal state node is colored red.
c. The walls are colored black.
d. Tiles on the optimal path are colored green.
e. Tiles that were explored but are not on the optimal path are colored blue.
f. Everything else (unexplored spaces) is colored white.

III. Algorithm

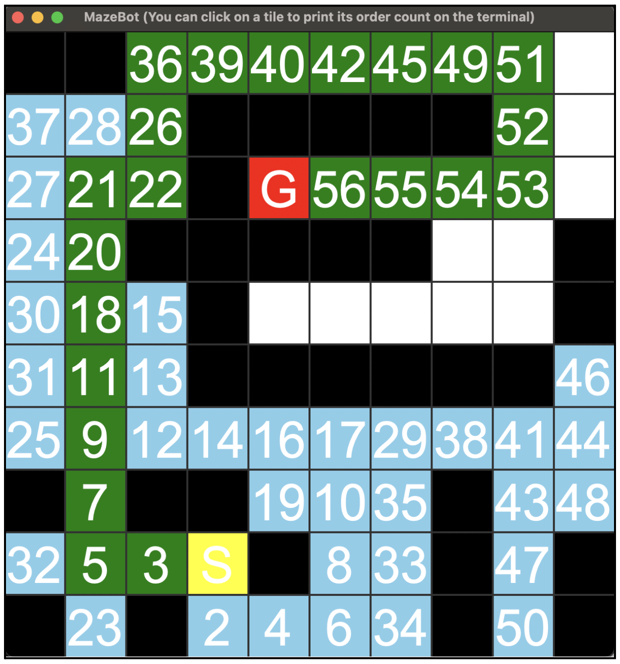
Pseudocode

```
// Run A*

Initialize priority queue (the frontier)
counter = 1

While frontier is not empty:
    Add start node to frontier with cost(0)
    current = pop from frontier
    current.explored = true
    current.order = counter
    counter++

    If current is goal:
        break

    For every valid move from current node:
        If node already in fronter:
            Update node’s cost to min(node.cost, current.cost + 1)
        Else:
            Push node to frontier with cost(current.cost + 1)

// Find the optimal path, if any path exists

If goal is explored:
    Initialize path
    current = goal
    While current node is not start:
        Add node to path
        current = the node that has the lowest cost among current's explored neighbors

    Add start to path
```

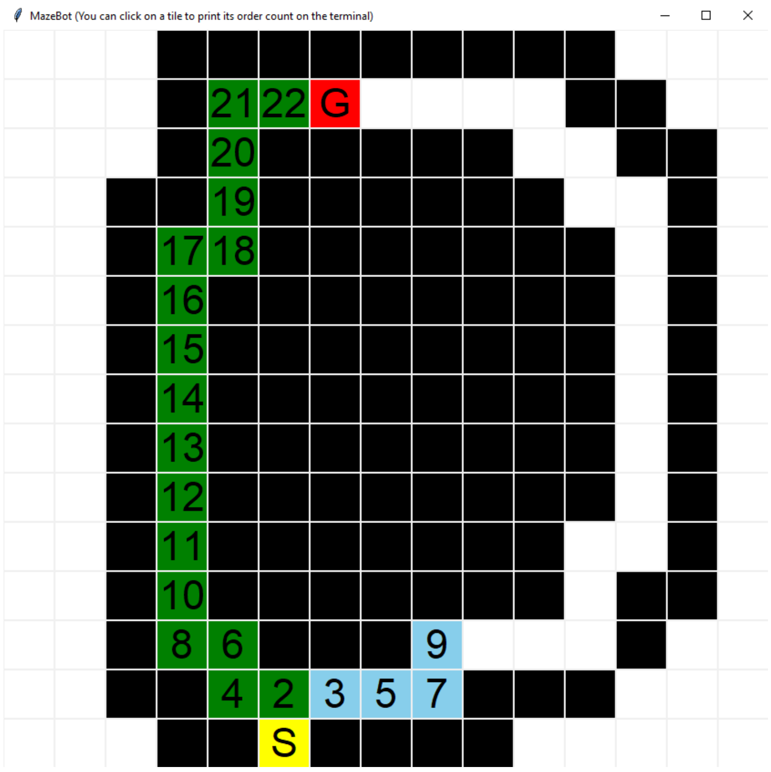
IV. Results and Analysis


MazeBot (You can click on a tile to print its order count on the terminal)

As seen in the figure beside, the maze bot can handle edge cases where the goal is unreachable from the start.


MazeBot (You can click on a tile to print its order count on the terminal)

For maze configurations with wall tiles, i.e. tiles that are inaccessible to the bot, the algorithm correctly identifies the optimal way to navigate the maze given these constraints. This is because tiles that are walls will never be added to the frontier list, meaning they will never even be considered as candidates for tiles on the optimal path.


MazeBot (You can click on a tile to print its order count on the terminal)
21
22
G
20
19
17
18
16
15
14
13
12
11
10
8
6
9
4
2
3
5
7
S

In cases where there are multiple paths to reach the goal state, the bot not only correctly finds the most optimal path (if any) among them, but finds the most optimal path first.

This is because the heuristic function is admissible. The Manhattan distance between 2 nodes is equivalent to the length of the shortest path between them assuming there are no walls on this path. If there are walls on this path, then the Manhattan distance underestimates the cost since additional steps are required to go around the walls.

V. Recommendations

The heuristic being used by the algorithm is not as accurate as it could be. This causes the bot to choose to explore less optimal paths first, leading to a slower overall runtime. The heuristic function would be better if it could penalize tiles that have walls on the theoretical path from it to the goal, as opposed to naively choosing based on Manhattan distance.

VI. Contributions of Each Group Member

ASTURIANO, Christian Emmanuel S.

Worked on the GUI, reading file input, and assisted with debugging the algorithm.

CHENG, Samuel Vincent T.

Worked on the implementation of A* search and the code segment that finds the optimal path.

CUSTER, Mark John T.

Worked on the report and documentation.

DE RAMOS, Ghrazielle Rei A.

Worked on the report and documentation.