**CAP4630 Introduction to Artificial Intelligence**

**Homework 3 (10 pts, Due Oct 2 2022)**

[Homework solutions must be submitted through Canvas. No email submission is accepted. Only pdf, word, txt and html files are allowed. If you have multiple pictures, please include all pictures in one Word/pdf file. You can always update your submissions before due date, but only the latest version will be graded.]

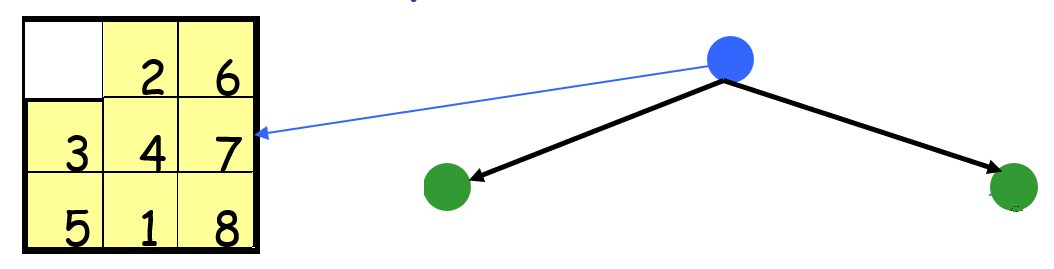


Figure 1: Eight-puzzle game: an initial state (left), and a portion of a search tree (right).

1. **[1 pt]** Figure 1 shows an initial state of the eight-puzzle game. Please create a search tree using this initial state. The search tree must have depth 2 (Figure 1 currently shows a search tree with depth 1) and shows all search nodes (including states corresponding to each search node.

Diagram, engineering drawing

Description automatically generated

1. **[1 pt]** When carrying out search process by using a search tree, will a finite state space always lead to a finite search tree? If not, please use eight-puzzle game in Figure 1 to show an example where a finite state space may lead to an infinite search tree.

No, a finite state space will not always lead to a finite state space. State spaces can be repeated in the search tree. (See states 1 & 8 in the figure above).

If the state or item being searched for is not present, the search tree will be infinite. If it is present and there’s a break statement, the search tree will be finite even with repeated states.

But a finite state space will NOT always lead to a finite search tree.

1. **[1 pt]** What are the four measures used to evaluate a search algorithm, please explain the definition of each measure [0.5 pt]? Please fill in the following table for the listed search algorithms [0.5 pt]
2. **Time Complexity –** The amount of time the search algorithm takes to complete.
3. **Space Complexity –** The amount of memory needed by the search algorithm.
4. **Optimality –** The minimum cost path is returned if a solution exists.
5. **Completeness –** The solution will be found if it exists.

**Use following symbols and O() notation for time and space complexities:**

*b*: branching factor; *c*: minimum step cost; *C\**: optimal path cost; *d*: depth of the shallowest solution; *m*: maximum depth of the search tree; *L*: the depth limit.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Performance Measure | Breath First Search | Depth First Search | Uniform Cost Search | Iterative Deeping Search | Depth-Limited Search |
| Complete? | Yes | No | Yes | Yes | Yes, if L >= d |
| Optimal? | Yes | No | Yes | Yes | No |
| Time Complexity | O(b^d) | O(b^m) | O(b^(C\*/c)) | O(b^d) | O(b^L) |
| Space Complexity | O(b^d) | O(bm) | O(b^(C\*/c)) | O(bd) | O(bL) |

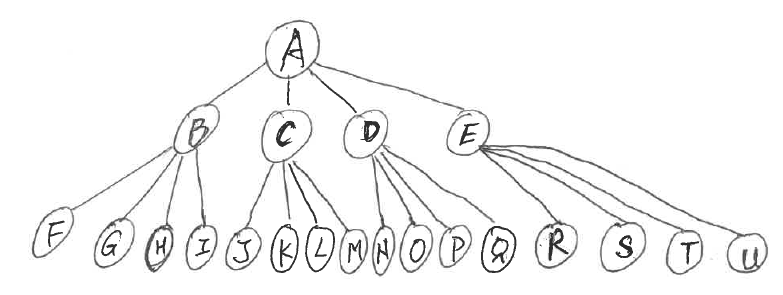


Figure 2

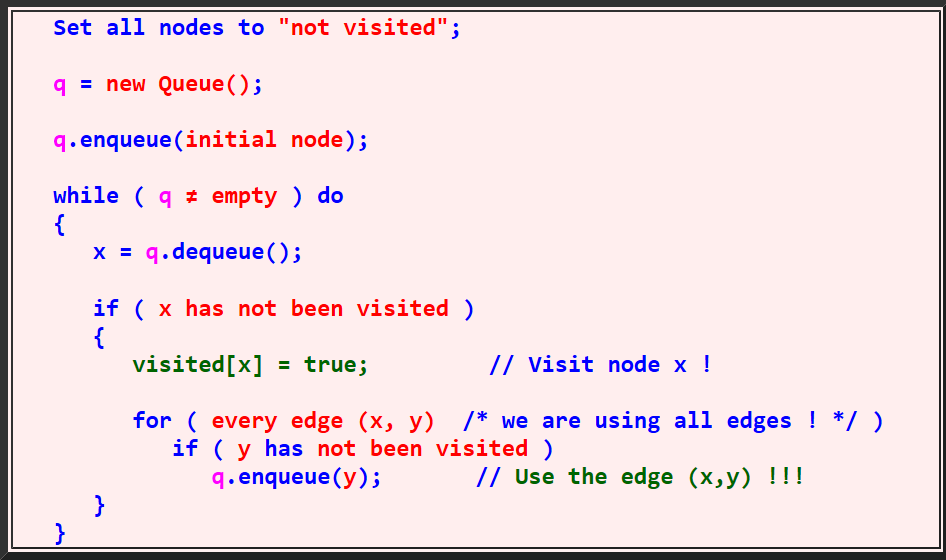


Figure 3: Breath First Search (BFS)

1. **[1 pt]** Figure 2 shows a search tree with 21 nodes. Please uses Breath First Search (BFS) algorithm showing in Figure 3 to report the order of the nodes being visited, and the Fringe structure (report fringe structure of each step as show in the table below) [0.5 pt]. Please compare the fringe structure and explain the memory consumption of each method, with respect to branching factor (b) and the search depth (d) [0.5 pt].

|  |  |  |  |
| --- | --- | --- | --- |
| Fringe | Node visited | Fringe Size | Search depth |
| A |  | 1 | 0 |
| BCDE | A | 4 | 1 |
| CDEFGHI | B | 7 | 2 |
| DEFGHIJKLM | C | 10 | 2 |
| EFGHIJKLMNOPQ | D | 13 | 2 |
| FGHIJKLMNOPQRSTU | E | 16 | 2 |

b (branching factor) = 4

At d (depth) = 0, the space complexity is b^d = 4^0 = O(1)

At d (depth) = 1, the space complexity is b^d = 4^1 = O(4)

At d (depth) = 2, the space complexity is b^d = 4^2 = O(16)

When these numbers are compared with the fringe structure (specifically its size), they hold true.

The fringe size would be growing by 4 each time a new node is visited because of the branching factor, but the fringe is also being reduced by 1 when the new node is removed. So the fringe is increasing by b-1 every time a new node is expanded. The final entry on the table is the last node to be expanded at search depth 2. You can see that the fringe has grown to the maximum space complexity of b^d = 16 here.

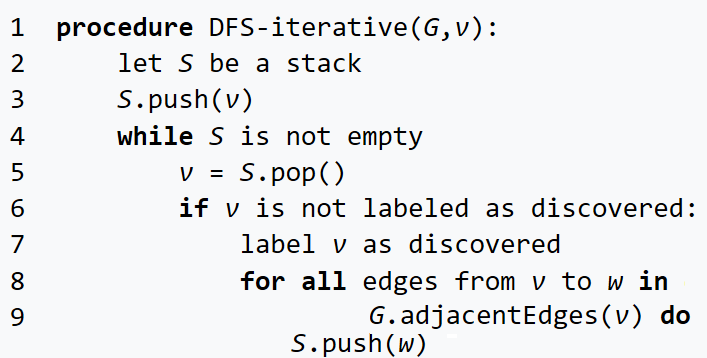


Figure 4 Depth First Search DFS (Method 1)

Text, letter

Description automatically generated

Figure 5 Depth First Search (Method 2)

1. **[1 pt]** Figure 2 shows a search tree with 21 nodes. Please uses Depth First Search algorithms showing in Figures 4 and 5 to report the order of the nodes being visited, and the Fringe structure of each method (report fringe structure of each step as show in the table below) [0.5 pt]. Please compare the fringe structure and explain the memory consumption of each method, with respect to branching factor (b) and the search depth (d) [0.5 pt].

DFS Method 1

|  |  |  |  |
| --- | --- | --- | --- |
| Fringe | Node Visited | Fringe Size | Search Depth |
| A | A | 1 | 0 |
| EDCB | B | 4 | 1 |
| EDCIHGF | F | 7 | 2 |
| EDCIHG | G | 6 | 2 |
| EDCIH | H | 5 | 2 |
| EDCI | I | 4 | 2 |
| EDC | C | 3 | 1 |
| EDMLKJ | J | 2 | 1 |
| EDMLK | K | 1 | 1 |
| EDML | L | 4 | 2 |
| EDM | M | 3 | 2 |
| ED | D | 2 | 2 |
| EQPON | N | 5 | 1 |
| EQPO | O | 4 | 2 |
| EQP | P | 3 | 2 |
| EQ | Q | 2 | 2 |
| E | E | 1 | 1 |
| UTSR | R | 4 | 2 |
| UTS | S | 3 | 2 |
| UT | T | 2 | 2 |
| U | U | 1 | 2 |

DFS Method 2

|  |  |  |  |
| --- | --- | --- | --- |
| Fringe | Node Visited | Fringe Size | Search Depth |
| A | A | 1 | 0 |
| AB | B | 2 | 1 |
| ABF | F | 3 | 2 |
| ABG | G | 3 | 2 |
| ABH | H | 3 | 2 |
| ABI | I | 3 | 2 |
| AC | C | 2 | 1 |
| ACJ | J | 3 | 2 |
| ACK | K | 3 | 2 |
| ACL | L | 3 | 2 |
| ACM | M | 3 | 2 |
| AD | D | 2 | 1 |
| ADN | N | 3 | 2 |
| ADO | O | 3 | 2 |
| ADP | P | 3 | 2 |
| ADQ | Q | 3 | 2 |
| AE | E | 2 | 1 |
| AER | R | 3 | 2 |
| AES | S | 3 | 2 |
| AET | T | 3 | 2 |
| AEU | U | 3 | 2 |

Fringe Structure and Memory Consumption

Both have the same space complexity because they have the same b, m and d. However, if you compare the fringe sizes of both methods, the first method approaches the upper bound of 8 with more frequency. The second method never increases above a space complexity of 3.

This means that the second method has a better space complexity.

b (branching factor) = 4, m (maximum depth) = 2

Space Complexity of DFS = O(b\*m) = 4\*2=O(8)

1. **[2 pts]** Figure 6 shows a state graph with eight nodes (where “S” is the starting node and “G” is the goal node).
   1. Starting from node “S”, please show a BFS search tree [0.25 pt], and a DFS search tree to find the goal node, respectively [0.25 pt].
   2. Explain whether BFS is optimal to find the goal node “G”, why or why not? [0.25 pt]
   3. Explain whether DFS is optimal to find the goal node “G”, why or why not? [0.25 pt]
   4. For the search trees build from the above step, please summarize the number of nodes generated, and summarize the time complexity (using O notation) of the BFS and DFS search, respectively (using *b* as branching factor, *d* as the shallowest goal node, and *m* as the search depth) [0.25 pt].
   5. For BFS search, explain how to avoid revisited states during the search process [0.25 pt]
   6. For DFS search, explain two ways to avoid revisited states during the search process [0.25 pt], explain memory consumption of each way of avoiding revisited states (using O notation) [0.25 pt]

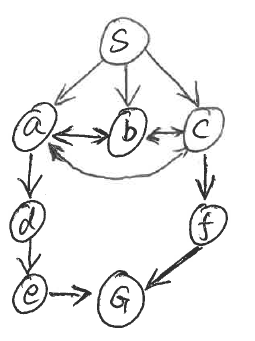


Figure 6 State Graph

***Please see the next two pages for my answers for #6***

**Letter

Description automatically generated**

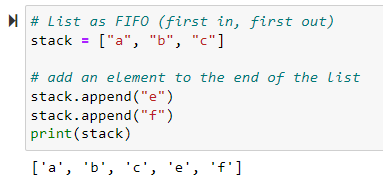
**A piece of paper with writing on it

Description automatically generated**

**For all programming tasks, please submit the Notebook as html or pdf files for grading (your submission must include scrips/code and the results of the script).**

If you are not familiar with Python programming (and want to use Python for the coding tasks), please check Python Plotting notebook and Python Simple Analysis notebook posted in the Canvas, before working the coding tasks.

For each subtask, please use task description (requirement) as comments, and report your coding and results in following format:



1. Coding problem [**1.5 pts**]. Using “node” class implemented in Homework 2 so implement following tasks:
2. Given an initial node, say (0,0), and a goal node, say (9,9), please create two nodes to denotes the initial and goal states. Design a 10x10 maze game board (with 100 locations), where 0 denotes open field and 1 denotes obstacle. [0.25 pt]
3. Design a successor function which takes a node and the maze game board layout as input, and returns successor(s) of the current node (the return values are a list of node(s)). The returned successor(s ) must satisfy the maze game board constraints. [0.25 pt]
4. Create a FIFO or a LIFO Fringe (you only need to implement one, not both), and insert initial node to the Frige. Then create a while loop, which continues as long as the Fringe is not empty
   1. Inside the while loop, pop out one node from the Fringe (one at a time), and use successor function to find its successors, then insert all successors to the Fringe. [0.25 pt]
   2. Record number of nodes in the Fringe with respect to the number of while loop repetitions. Draw the plot (x-axis shows number of repetitions, and y-axis shows number of nodes). Explain the trend of the growth patterns (e.g. .linear, exponential etc.) [0.25 pt]

**The growth of my implementation is linear, as will be displayed in a graph later. This is because the implementation is Depth First Search.**

* 1. Each time pop out a node from the Fringe, check whether the node is the goal node. Return “Goal” if finds the goal, or continue otherwise [0.25 pt]

1. Explain why or why not the above BFS or a DFS algorithm may find the goal node? What are the main disadvantage of the above implementation [0.25 pt]

**Yes, this implementation will definitively find the goal node. This is true as long as the goal node position remains (9,9). This is true because of the placement of the order of my checking of adjacent squares.**

**If I search for a different goal node, (i.e. change the position to let’s say (0,9)) the space complexity of the algorithm becomes so large that it will fill up memory and crash my browser. This is because this implementation does not check to see if a node has already been visited. This causes way too many duplicate nodes to be added to the fringe. Theoretically, it may eventually find the goal node, but the space complexity increases too fast for at least my computer to make the computation.**

# Yes this implementation will find the goal node. This is because of my placement of the order of my

# checking of adjacent squares and the coordinates I'm searching for. The main disadvantage is that this implementation

# doesn't check to see if a successor has been previously visited. This can add a lot of duplicates

# to the fringe.

Text

Description automatically generatedText

Description automatically generatedText

Description automatically generatedChart, line chart

Description automatically generated

1. [**1.5 pts**] The “**BlindSearchMaze [Notebook, html]**” posted on Canvas shows a Maze game using DFS and BFS search (using “DFS” or “BFS” parameters). Use Notebook as the skeleton code, validate and compare following settings and results.
   1. Using Figure 7 as the game field, set initial state as [0, 0] and goal state as [0, 1]. Report number of nodes expanded by BFS and DFS, respectively. Report final path discovered by each algorithm, respectively [0.25 pt]. Explain why or why not the method is optimal, and why one approach expands far more nodes than the other method [0.25 pt]
   2. Set initial state as [0, 0] and goal state as [0, 1], [0, 2], [0, 3], [0, 5], [0, 6], [0, 7], [0, 8], [0, 9], respectively. Create one plot to show the number of maximum Fringe size of BFS (x-axis denotes the goal state, and y-axis denotes the maximum Fringe size) [0.25 pt]. Create one plot to show the number of maximum Fringe size of DFS (x-axis denotes the goal state, and y-axis denotes the maximum Fringe size) [0.25 pt]. Explain how the Fringe size grows with respect to the search depth for DFS and BFS, respectively. [0.25 pt]
   3. For goal state [0, 9], compare path returned by BFS and DFS, respectively. Which method optimal, and which method is not optimal. Why? [0.25 pt]

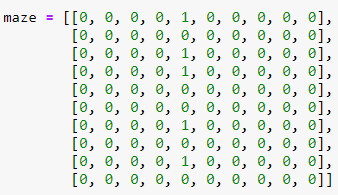
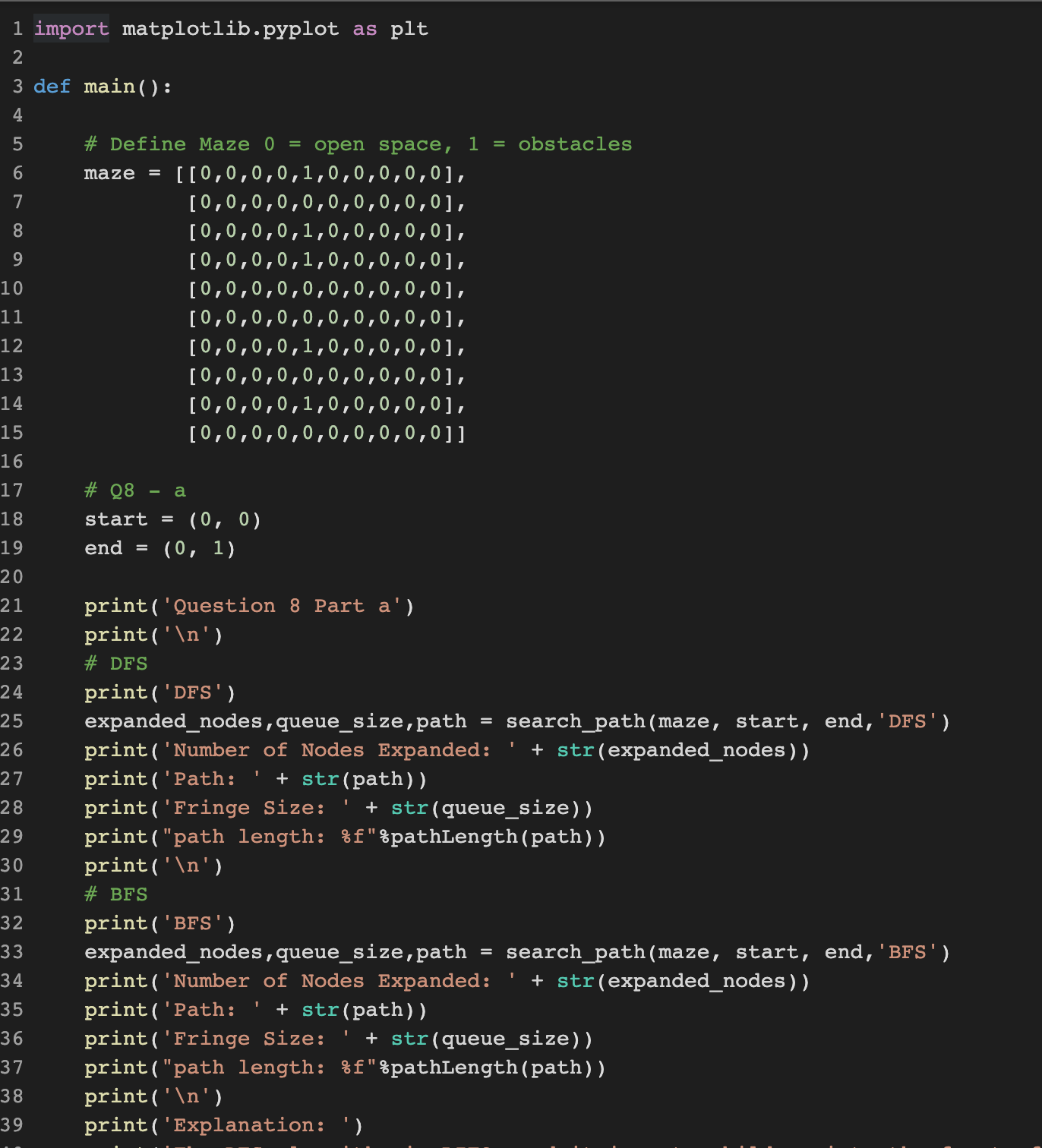
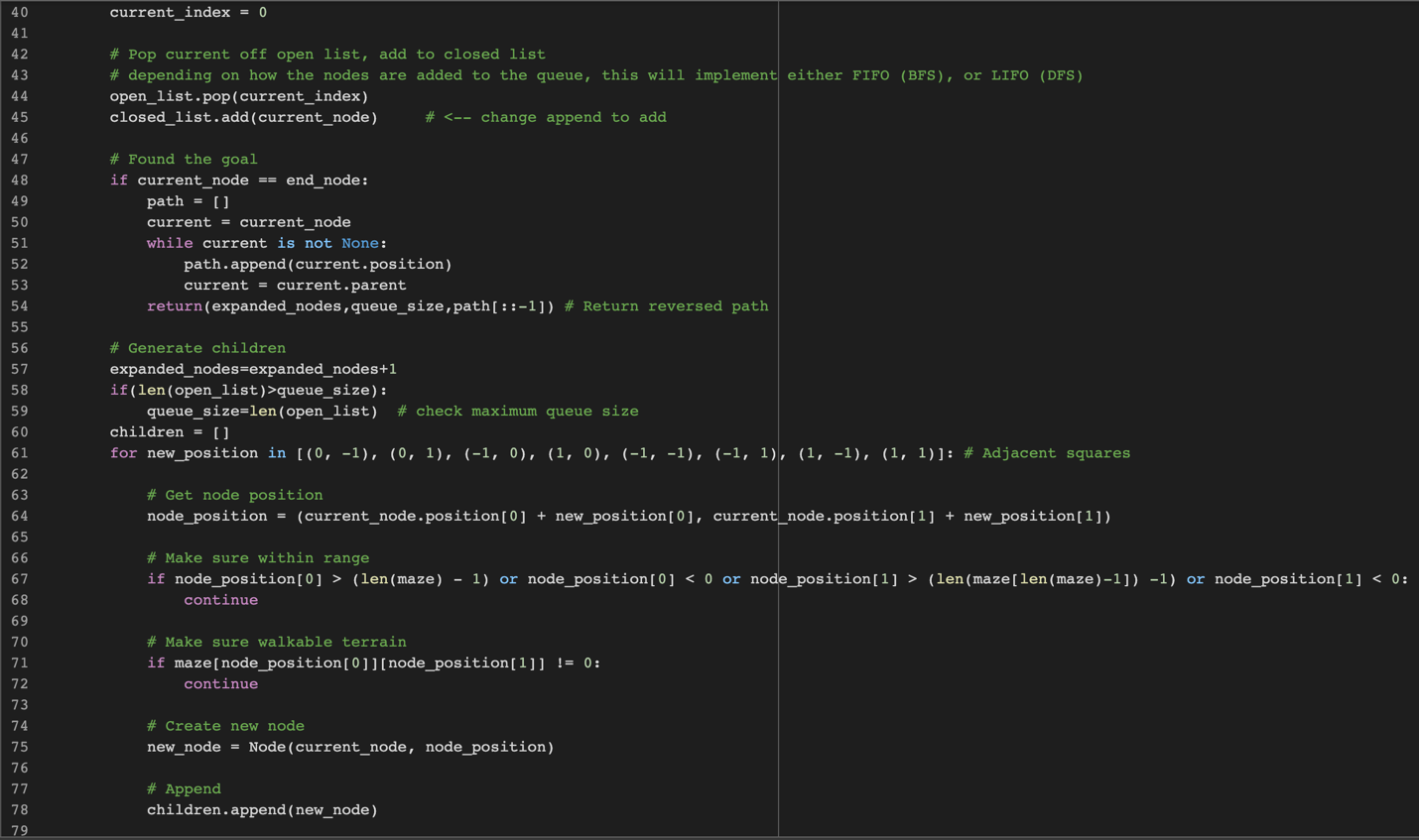
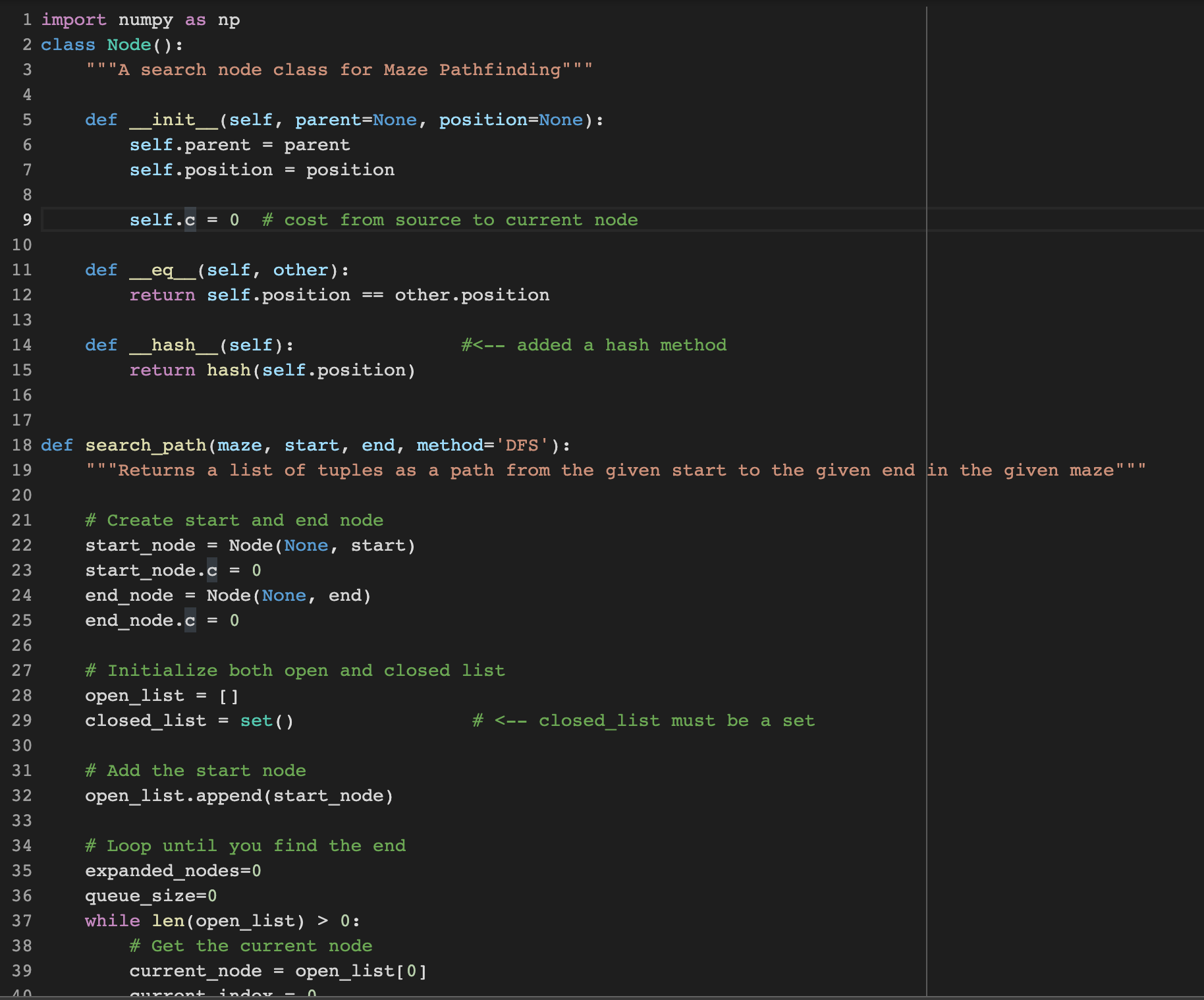


Figure 7: Maze game field

***My explanations, objectives and requirements for all of these are included at the end of the code. I embedded print statements inside the code to display all my answers to these questions. The last 2 images contain all the explanations.***

Text

Description automatically generatedText

Description automatically generatedChart, line chart

Description automatically generatedText

Description automatically generated