

**Final Report: Nurse Robot**

Adrienne Loc and Parisa Aghazadeh

Saint Joseph's University, College of Arts and Sciences

CSC 362: Artificial Intelligence

Dr. Babak Forouraghi

May 7, 2024

## Contents

<b>Final Report: Nurse Robot</b> .....	1
Introduction.....	3
Running the Robot .....	3
Map Representation .....	4
Map.py .....	5
LocationChoice.py .....	5
Path Finding .....	5
Conclusion .....	6
Ranking Statement .....	8
Appendix A.....	9
Appendix B .....	10
Appendix C .....	11
Appendix D.....	12
Appendix E .....	13
Appendix F.....	14
Appendix G.....	15
Appendix H.....	16
Appendix I .....	17

## Introduction

Artificial intelligence is becoming an integral part of today's society. From chatbots like ChatGPT to self-driving cars, AI is being blended into our daily lives. In this project, we created a nurse robot that delivers medication to the specified locations with the locations being ordered by their priorities. The goal of this project was to not only explore the algorithms used in AI path finding, such as Dijkstra's algorithm and A\* algorithm, but to also ease the burden on nurses so that they no longer have to travel to retrieve medications. In this report, we will cover the key features and details of this robot and the differences we noticed between the two path finding algorithms that we used.

## Running the Robot

When the program is run the user is greeted with a window that displays the hospital map. The hospital map is colored by priority rather than the wards. The priority colors are as follows: blue has a priority of 1, green has a priority of 2, yellow has a priority of 3, orange has a priority of 4, and red has a priority of 5. At the bottom of the window, there is a button that the user can click to select a text file for the program to read ([see Appendix A](#)). This button makes it so that, even when the program finds a path for a singular input, rather than terminating, the user can click the button again to find a path for another starting location and list of goal locations. This lets the user find as many paths as they would like without having to rerun the program. The text file used as the input should contain the algorithm the user would like to use on the first line (A\* or Dijkstra's), the starting coordinate on the second line, and the list of goal coordinates on the last ([see Appendix B](#)). Once the user selects a text file that contains all the information, the program will draw a path from the start position to all the goal states and output a "SUCCESS" message when it has found a path. Something notable about the path is that the user can not only

see the path being drawn, but the path also changes color whenever it reaches a goal state so that the user can see the different paths being drawn ([see Appendix C](#)). If the user's starting position is outside of the hospital itself, robot will still find a path to the goal locations due to the gaps in the map that represent doors ([see Appendix D](#)). However, if the robot is unable to find a path at all (the robot is trapped for instance), then the program will not draw a path and will output a "FAIL" message ([see Appendix E](#)). Additionally, if the list of goal locations contains a location that is unreachable, the program will skip over that coordinate and focus on the ones that it can reach ([see Appendix F](#)).

### **Map Representation**

The map that our program generates uses a binarized image of the map outline. As a result, we were able to generate a highly detailed representation of the hospital map. However, because it was so detailed, it made it difficult to identify locations by their coordinates on the map representation itself. To go through every single square and figure out what its coordinates are would have been time-consuming and tedious, and then we would have to consider the chances of us making a mistake. To avoid that issue we decided to create two additional programs that would help us with map representation: Map.py and LocationChoice.py.

In both Map.py and LocationChoice.py, it essentially takes keyboard input from the user when they select a cell and allows the user to move up and down. When the user moves, it will highlight where the user is moving. In the following two sections, we describe the helper programs.

## Map.py

The Map.py helper program was what we used to help us select and assign priorities to specific coordinates. It can select multiple cells in bulk if you draw a box by highlighting all the cells within the box ([see Appendix G](#)). Once we highlighted the cells we wanted for a specific priority, we would type in the priority and click the print button to have it print out all the coordinates followed by the priority ([see Appendix H](#)). The program keeps track of what cell has been selected by storing the selected cells in a list if that cell is not already in the list. Then we copied and pasted the output into the prio text file.

## LocationChoice.py

The LocationChoice.py helper program is similar to the Map.py in terms of functionality. We can select cells and move around on the map. The difference is in the print functionality. The print functionality only prints the cells that have been clicked rather than priority ([see Appendix I](#)). We used this helper program to help us pick coordinates to use in our test text files.

## Path Finding

The goal of this robot was to find the most optimum path given a starting location and a list of goal locations that the robot must reach. The robot that we created has a heuristic that it follows which is the Manhattan distance between the robot's current state and the goal state. The two algorithms that the robot used to successfully generate a path to the multiple goal states are: A\* algorithm and Dijkstra's algorithm. As mentioned before, one of the goals of this project was to explore the differences between how the two algorithms performed. Dijkstra's algorithm does not consider the heuristic whereas the A\* algorithm does. This means that the path that Dijkstra's

algorithm generates is based solely on the path cost itself. This slight difference does change the path the robot takes to reach a goal destination.



**Figure 1: A\* vs Dijkstra's**

In Figure 1 we can see how the paths of A\* and Dijkstra's algorithm diverge from each other. Though both paths are going to the same locations in the same order, Dijkstra's algorithm results in a square being drawn in hallway the map whereas A\* does not have a square. This is because, as mentioned previously, Dijkstra does not take into consideration the heuristic. This means that the square that resulted from Dijkstra's algorithm was because the algorithm solely focused on path cost and picked the cells that was closest to where the robot was (meaning the cells with the lowest path cost)

## Conclusion

In this project, we were able to successfully create a simulation of a robot nurse that is able to deliver medication using both the A\* algorithm and Dijkstra's algorithm. Since we decided to implement both algorithms, we were able to explore the differences between the two

algorithms and analyze how lack of consideration for the heuristic affected the path itself as seen in Figure 1. It was intriguing to see how the path changed based on what algorithm we decided to use. Furthermore, besides exploring the algorithms and the differences between them, we were also able to learn and examine the capabilities of tkinter and map representation in programs which was entertaining and fulfilling.

## Ranking Statement

### Member 1: Adrienne Loc

My teammate and I agreed that I handled 50% of the overall project. My specific tasks included:

- Dijkstra's algorithm
- Request Processing
- Map creation
- Research about keyboard input through tkinter
- Map program: cell selection, bounding box, cell printing
- Location choice program: cell selection, cell printing

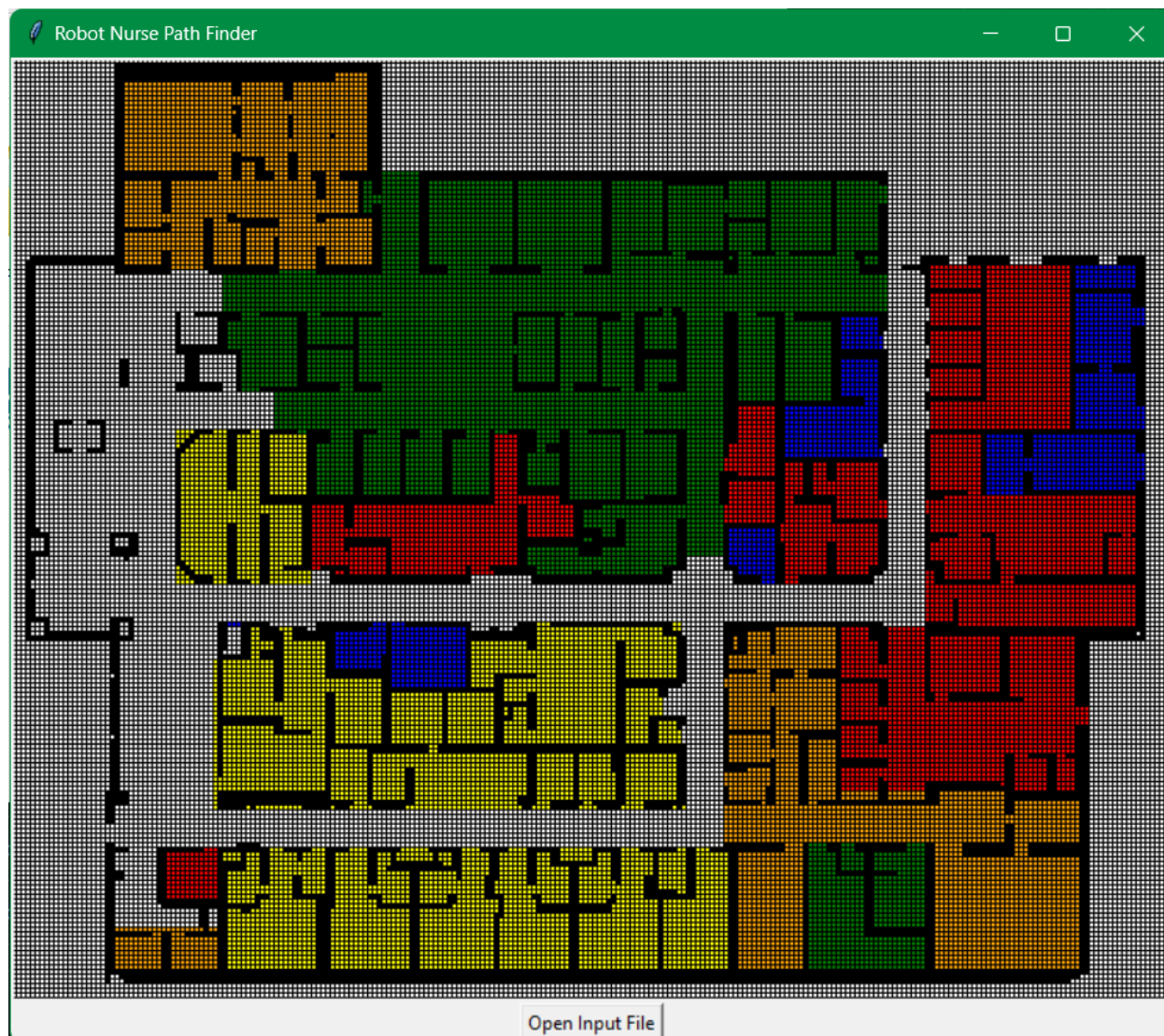
### Member 2: Parisa Aghazadeh

My teammate and I agreed that I handled 50% of the overall project. My specific tasks included:

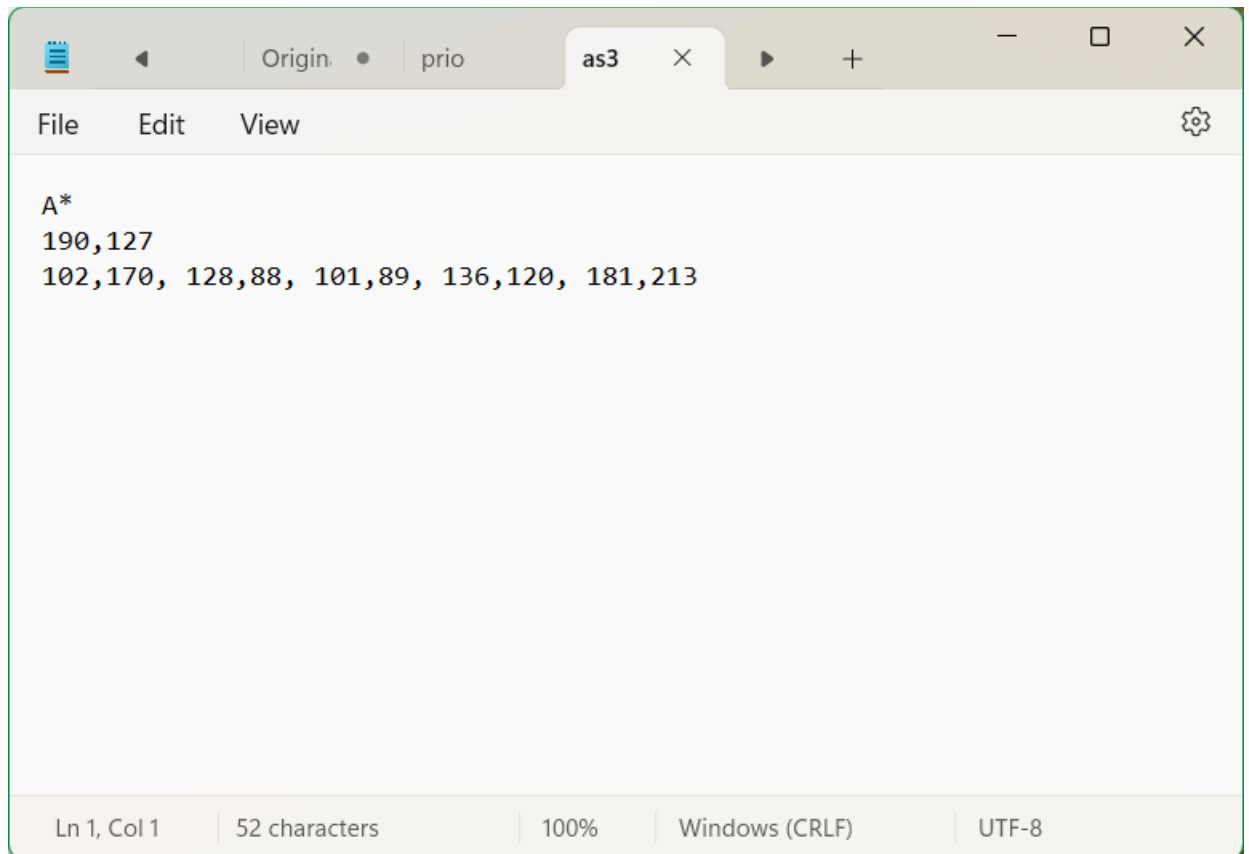
- A\* algorithm
- Path animation
- Open file function
- Retrieving neighbors
- Research about keyboard input through tkinter
- Map program: map generation, keyboard input, clear button
- Location choice program: keyboard input



## Appendix A



## Appendix B

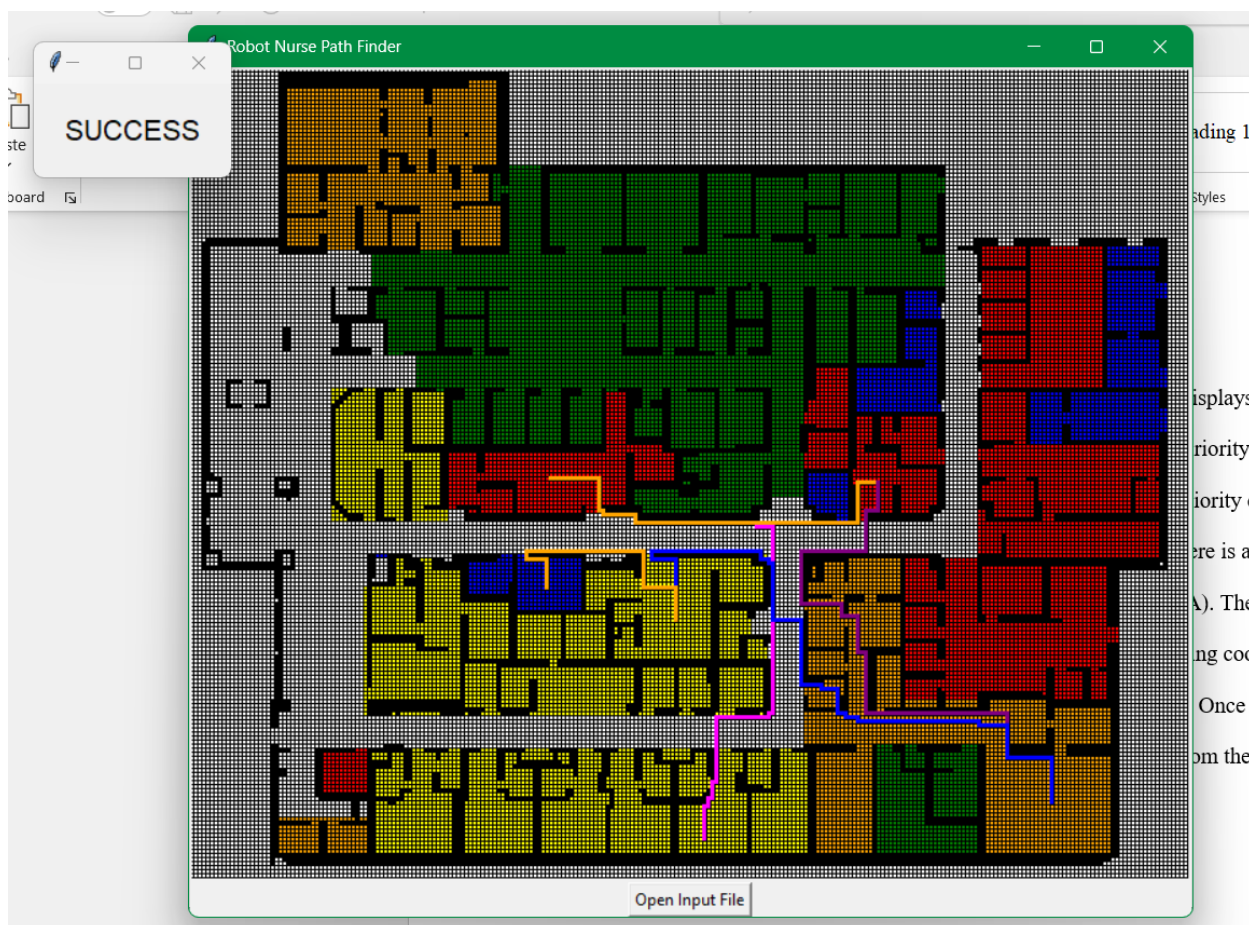


The image shows a screenshot of a code editor window. The window has a title bar with a file icon, a back arrow, and a tab labeled 'as3'. Below the title bar is a menu bar with 'File', 'Edit', and 'View' options. The main area of the editor contains the following text:

```
A*
190,127
102,170, 128,88, 101,89, 136,120, 181,213
```

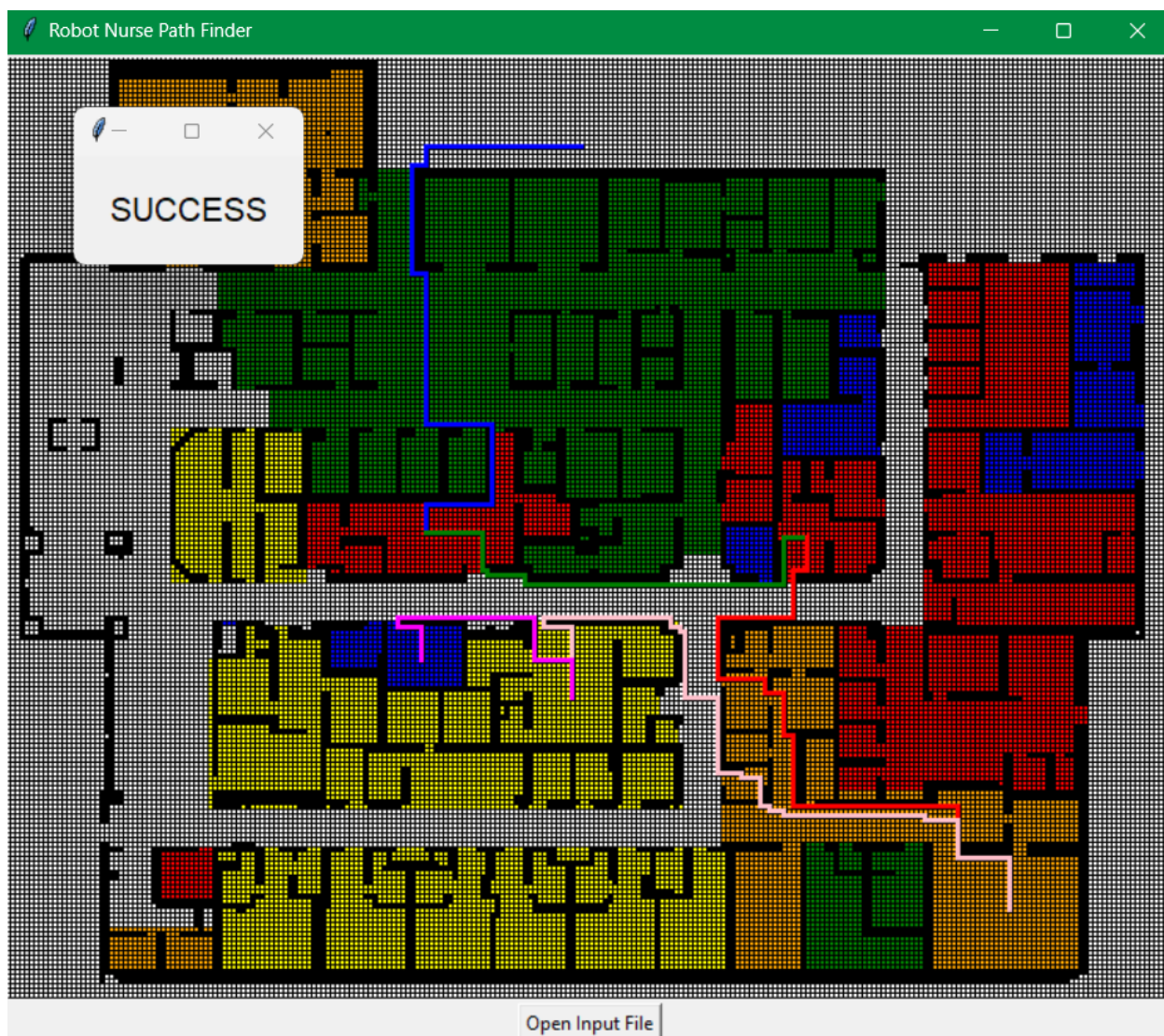
At the bottom of the window, there is a status bar with the following information: 'Ln 1, Col 1', '52 characters', '100%', 'Windows (CRLF)', and 'UTF-8'.

## Appendix C

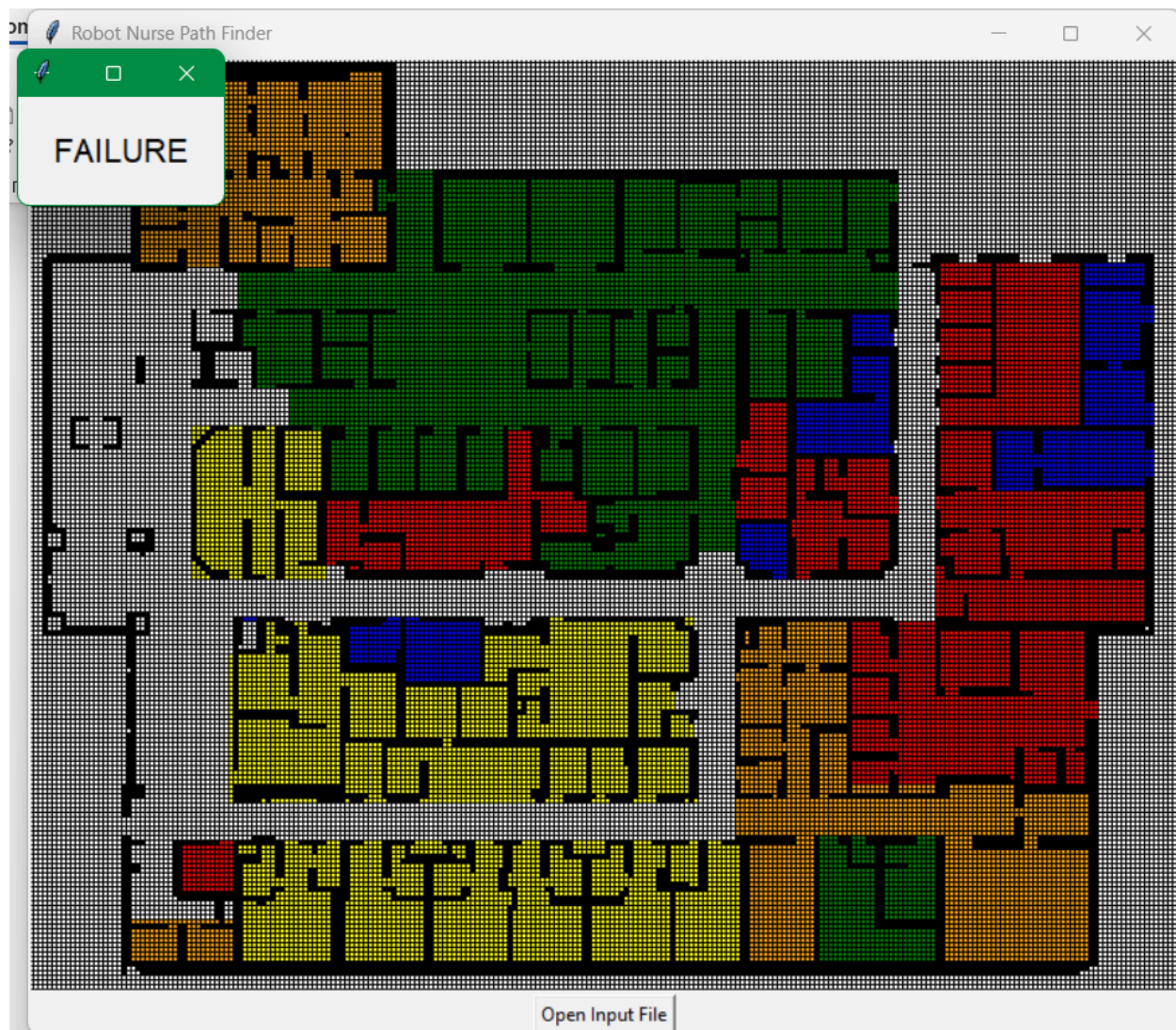




## Appendix D



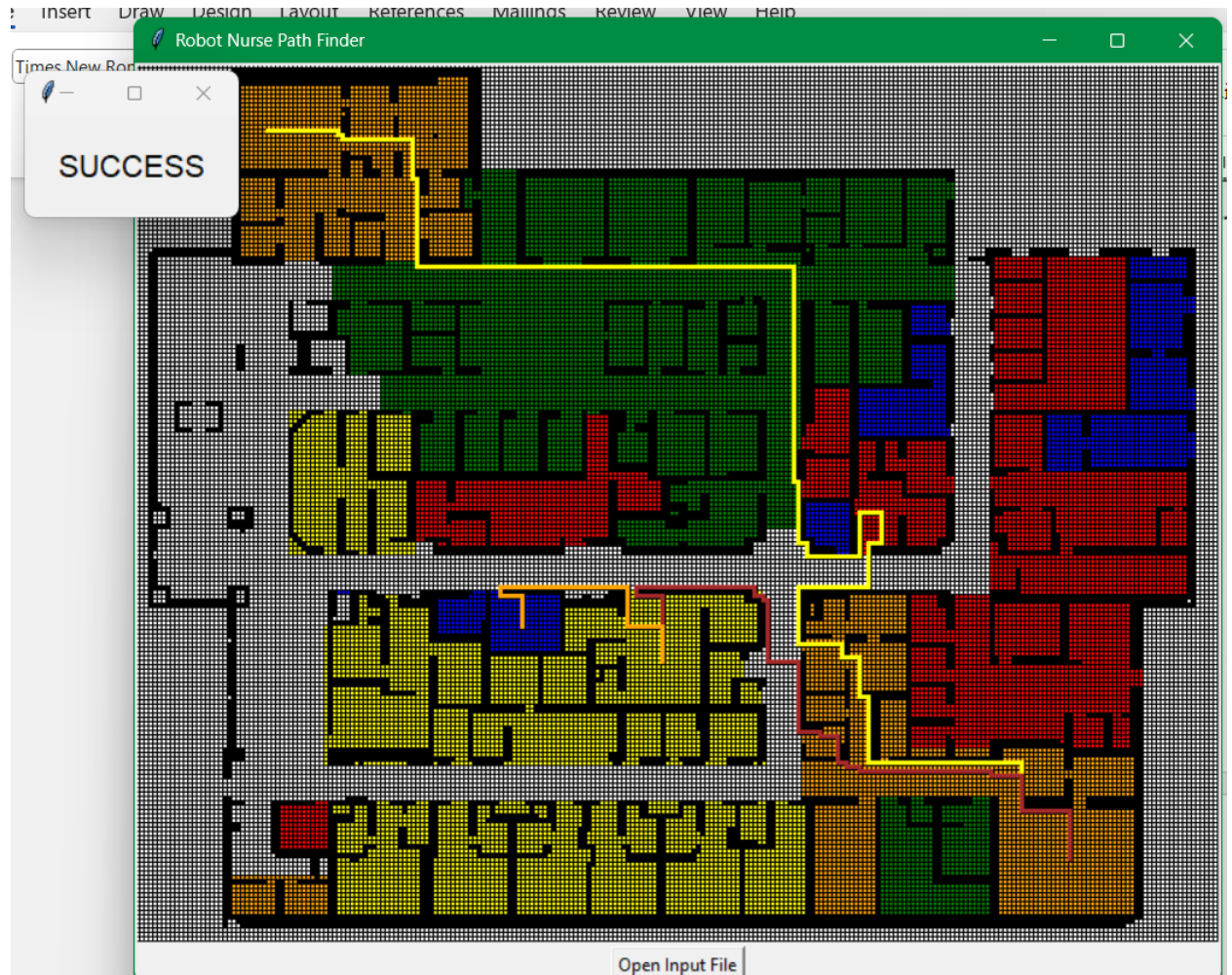
## Appendix E



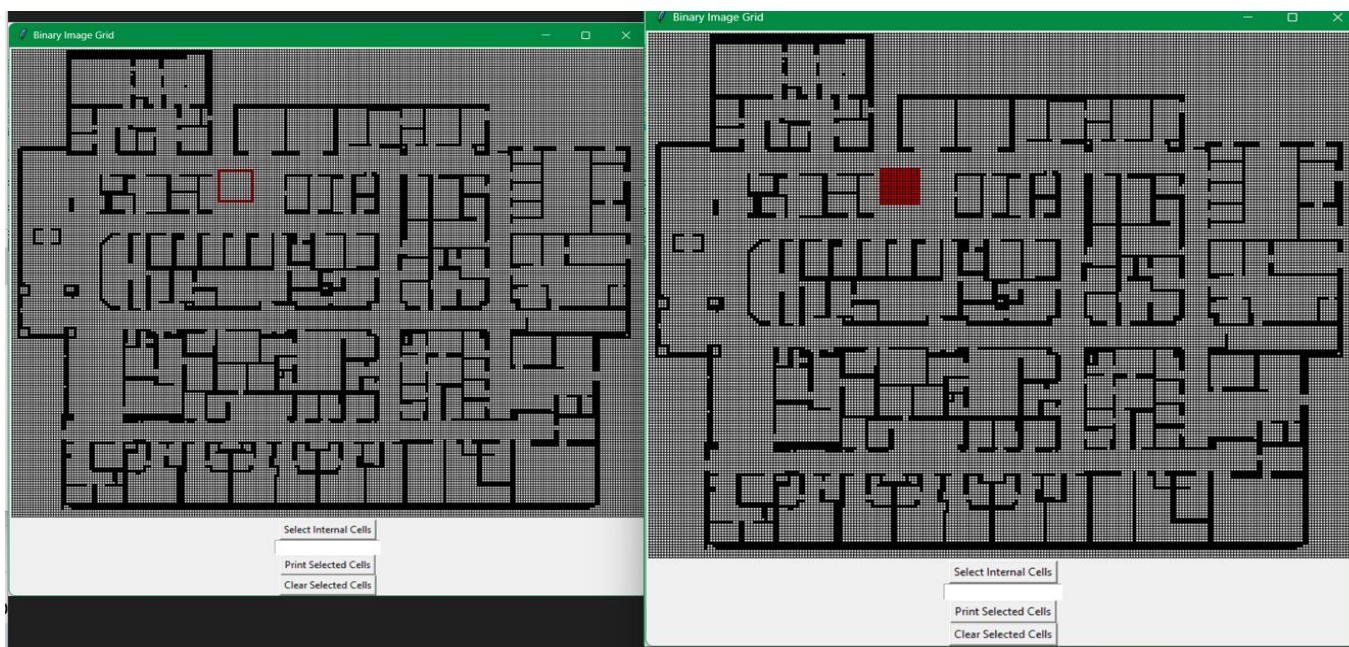


## Appendix F

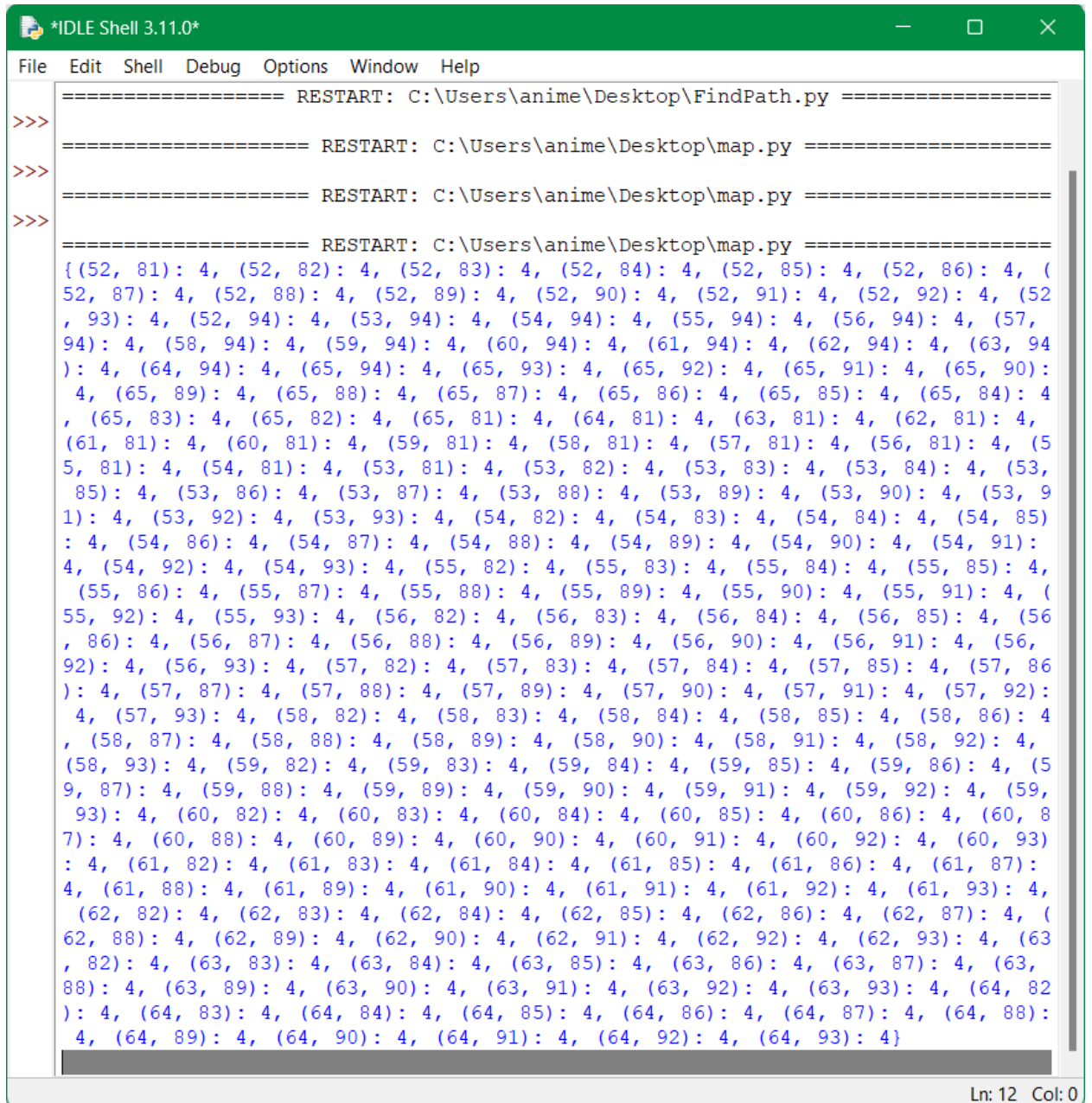
The input file for this path had multiple locations that were unreachable.



## Appendix G



## Appendix H



```

===== RESTART: C:\Users\anime\Desktop\FindPath.py =====
>>>
===== RESTART: C:\Users\anime\Desktop\map.py =====
>>>
===== RESTART: C:\Users\anime\Desktop\map.py =====
{(52, 81): 4, (52, 82): 4, (52, 83): 4, (52, 84): 4, (52, 85): 4, (52, 86): 4, (
52, 87): 4, (52, 88): 4, (52, 89): 4, (52, 90): 4, (52, 91): 4, (52, 92): 4, (52
, 93): 4, (52, 94): 4, (53, 94): 4, (54, 94): 4, (55, 94): 4, (56, 94): 4, (57,
94): 4, (58, 94): 4, (59, 94): 4, (60, 94): 4, (61, 94): 4, (62, 94): 4, (63, 94
): 4, (64, 94): 4, (65, 94): 4, (65, 93): 4, (65, 92): 4, (65, 91): 4, (65, 90):
4, (65, 89): 4, (65, 88): 4, (65, 87): 4, (65, 86): 4, (65, 85): 4, (65, 84): 4
, (65, 83): 4, (65, 82): 4, (65, 81): 4, (64, 81): 4, (63, 81): 4, (62, 81): 4,
(61, 81): 4, (60, 81): 4, (59, 81): 4, (58, 81): 4, (57, 81): 4, (56, 81): 4, (5
5, 81): 4, (54, 81): 4, (53, 81): 4, (53, 82): 4, (53, 83): 4, (53, 84): 4, (53,
85): 4, (53, 86): 4, (53, 87): 4, (53, 88): 4, (53, 89): 4, (53, 90): 4, (53, 9
1): 4, (53, 92): 4, (53, 93): 4, (54, 82): 4, (54, 83): 4, (54, 84): 4, (54, 85)
: 4, (54, 86): 4, (54, 87): 4, (54, 88): 4, (54, 89): 4, (54, 90): 4, (54, 91):
4, (54, 92): 4, (54, 93): 4, (55, 82): 4, (55, 83): 4, (55, 84): 4, (55, 85): 4,
(55, 86): 4, (55, 87): 4, (55, 88): 4, (55, 89): 4, (55, 90): 4, (55, 91): 4, (
55, 92): 4, (55, 93): 4, (56, 82): 4, (56, 83): 4, (56, 84): 4, (56, 85): 4, (56
, 86): 4, (56, 87): 4, (56, 88): 4, (56, 89): 4, (56, 90): 4, (56, 91): 4, (56,
92): 4, (56, 93): 4, (57, 82): 4, (57, 83): 4, (57, 84): 4, (57, 85): 4, (57, 86
): 4, (57, 87): 4, (57, 88): 4, (57, 89): 4, (57, 90): 4, (57, 91): 4, (57, 92):
4, (57, 93): 4, (58, 82): 4, (58, 83): 4, (58, 84): 4, (58, 85): 4, (58, 86): 4
, (58, 87): 4, (58, 88): 4, (58, 89): 4, (58, 90): 4, (58, 91): 4, (58, 92): 4,
(58, 93): 4, (59, 82): 4, (59, 83): 4, (59, 84): 4, (59, 85): 4, (59, 86): 4, (5
9, 87): 4, (59, 88): 4, (59, 89): 4, (59, 90): 4, (59, 91): 4, (59, 92): 4, (59,
93): 4, (60, 82): 4, (60, 83): 4, (60, 84): 4, (60, 85): 4, (60, 86): 4, (60, 8
7): 4, (60, 88): 4, (60, 89): 4, (60, 90): 4, (60, 91): 4, (60, 92): 4, (60, 93)
: 4, (61, 82): 4, (61, 83): 4, (61, 84): 4, (61, 85): 4, (61, 86): 4, (61, 87):
4, (61, 88): 4, (61, 89): 4, (61, 90): 4, (61, 91): 4, (61, 92): 4, (61, 93): 4,
(62, 82): 4, (62, 83): 4, (62, 84): 4, (62, 85): 4, (62, 86): 4, (62, 87): 4, (
62, 88): 4, (62, 89): 4, (62, 90): 4, (62, 91): 4, (62, 92): 4, (62, 93): 4, (63
, 82): 4, (63, 83): 4, (63, 84): 4, (63, 85): 4, (63, 86): 4, (63, 87): 4, (63,
88): 4, (63, 89): 4, (63, 90): 4, (63, 91): 4, (63, 92): 4, (63, 93): 4, (64, 82
): 4, (64, 83): 4, (64, 84): 4, (64, 85): 4, (64, 86): 4, (64, 87): 4, (64, 88):
4, (64, 89): 4, (64, 90): 4, (64, 91): 4, (64, 92): 4, (64, 93): 4}

```

Ln: 12 Col: 0



## Appendix I

