

# Final Iteration: Autonomous Delivery Robot Navigation

Team Name: JanVas

November 18, 2024

## 1 Project Scope

The aim of this project is to design and develop a navigation algorithm for an autonomous robot/vehicle that must reach way points while avoiding obstacles. If time allows, this algorithm will include EV charging stations along the route and optimize a path accordingly. The project's main objectives are:

- To allow delivery vehicle to most efficiently deliver packages.
- To enable robot to autonomously navigate obstacles.
- To develop a robust backtracking algorithm that can be applied to other fields.
- If time allows, expand into greedy/graph algorithms by incorporating the car fueling along its path.
- If time allows, incorporate the outputs onto a UI on a front-end web page.

The expected outcomes include a functioning web-based system, proper documentation, and a final project report.

## 2 Project Plan

### 2.1 Timeline

The overall timeline for the project is divided into phases:

- **Week 1 (October 7 - October 13):** Define project scope, establish team roles, and outline skills/tools.
- **Week 2 (October 14 - October 20):** Begin development, set up the project repository, and start coding basic system functionalities.
- **Week 3 (October 21 - October 27):** Continue coding, work on back-end integration, start front-end code, and start writing technical documentation.
- **Week 4 & 5 (October 28 - November 10):** Complete the back-end and front-end, then integrate front-end elements. Begin PowerPoint presentation.
- **Week 6 (November 11 - November 17):** Finalize the system, conduct testing, and continue with the report.
- **Week 7 (November 18 - November 24):** Revise and finalize the technical report and the PowerPoint presentation.
- **Week 8 (November 25 - November 28):** Final presentation, report submission, and project closure.

## 2.2 Milestones

Key milestones include:

- Project Scope and Plan (October 7).
- GitHub Repository Setup and Initial Development (October 9).
- Back-end Completion (October 28).
- Final System Testing and Report Draft (November 10).
- Final Presentation and Report Submission (November 28).

## 2.3 Team Roles

- **Vasishta Malisetty:** Responsible for front-end design and user interface development.
- **Jani Passas:** Back-end developer, focusing on the database and server-side logic.

These roles are flexible and may change as the project progresses.

# 3 Team Discussion Summary (October 7)

## 3.1 Skills Assessment

Team members need the following skills to complete the project:

- Proficiency in HTML/CSS and JavaScript for front-end development.
- Back-end skills in C++/Crow for the server-side logic.
- Familiarity with GitHub for version control.
- Use of Overleaf for writing the final technical report.

## 3.2 Tools & Technologies

- **Programming Languages:** C++ (Crow), HTML and CSS.
- **Database:** MongoDB.
- **Collaboration Tools:** GitHub for version control, Overleaf for report writing.

## 3.3 Team Responsibilities

Each team member will focus on specific areas, but roles may adapt as the project progresses:

- Front-end Development.
- Back-end Development.
- Technical Documentation.
- System Testing.

# 4 Skills & Tools Assessment (October 8)

## 4.1 Skills Gaps & Resource Plan

We identified a few gaps in front-end programming and database management. To address these, we plan to attend workshops and follow tutorials during Week 2-3.

## 4.2 Tools

We will use the following tools:

- C++ (Crow) for back-end development.
- MongoDB for database management.
- GitHub for version control and collaboration.
- Overleaf for the technical report.

## 5 Initial Setup Evidence (October 9)

### 5.1 Project Repository

The project repository has been created on GitHub, accessible by all team members. The repository can be found at <https://github.com/Jani-Passas/AlgoProjects>.

### 5.2 Setup Proof

The development environment has been successfully set up. Screenshots of the setup process are provided in the Appendix.

## 6 Progress Review (October 10)

### 6.1 Progress Update

The team successfully completed the initial setup of the development environment and repository.

### 6.2 Issues Encountered

We encountered issues with creating branches within the repository, but the team plans to resolve this by consulting documentation and tutorials over the weekend.

## 7 Revised Project Plan (October 10)

### 7.1 Updated Plan

After reviewing our progress, we updated the timeline to allow for additional time to address back-end integration challenges. The milestone for back-end completion is now moved to November 10.

### 7.2 Justification for Changes

The adjustment was necessary due to unforeseen technical challenges in integrating the front-end with the back-end. We plan to spend extra hours on resolving these issues to meet the project deadline.

## 8 Algorithm Design (October 27)

### 8.1 Description

For this project, we decided to go with an backtracking algorithm as it is a robust algorithm to find the shortest, most optimal path for the EV robot to navigate. Current status has an input of a square maze where 1 represents the path options, then the algorithm will print out a graph/matrix of the same size with 1's in the spots that the robot found the optimal path.

## 8.2 Data Representation

Backtracking is necessary as the constraints of the path include avoiding obstacles, finding EV charging stations when the robot is low on power, and completing a set amount of deliveries within a day. We created a block diagram representation for our algorithm in order to clearly communicate and visualize the individual components needed for the software implementation of this project. Note that currently all the outputs for the backend code can be displayed in the terminal, but as scripts get combined and expanded on, a front-end display and visualizations will be made to better show outputs.

## 8.3 Detailed Steps

Step 1: Represent the environment with a Grid system

Step 2: Define valid movements of the robot (Forward, Backward, Left, Right, Diagonals)

Step 3: Implement Back Tracking Functionality

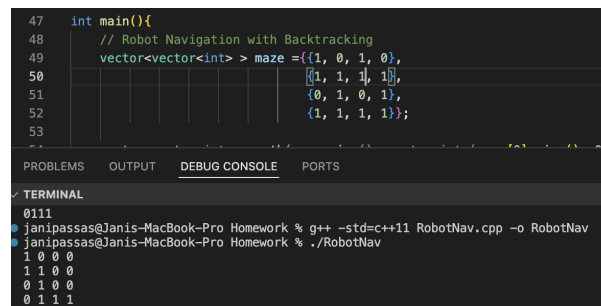
Step 4: Prune the brute force solutions until you get an optimal solution

Step 5: Handle any edge cases (i.e. the obstacles move, or the robot needs to go to a charging station).

## 9 Testing Results and Challenges Faced (October 27)

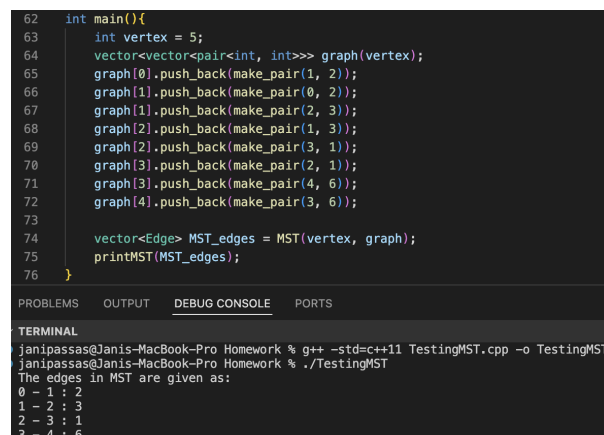
### 9.1 Testing Results

After reviewing our progress, we updated the timeline to allow for additional time to address back-end integration challenges. The milestone for back-end completion is now moved to November 10.



```
47 int main(){
48     // Robot Navigation with Backtracking
49     vector<vector<int>> maze ={{1, 0, 1, 0},
50                               {{1, 1, 1, 1}},
51                               {{0, 1, 0, 1}},
52                               {{1, 1, 1, 1}}};
53
PROBLEMS OUTPUT DEBUG CONSOLE PORTS
TERMINAL
0111
janipassas@Janis-MacBook-Pro Homework % g++ -std=c++11 RobotNav.cpp -o RobotNav
janipassas@Janis-MacBook-Pro Homework % ./RobotNav
1 0 0 0
1 1 0 0
0 1 0 0
0 1 1 1
```

Figure 1: Backend Output of Backtracking Algorithm



```
62 int main(){
63     int vertex = 5;
64     vector<vector<pair<int, int>>> graph(vertex);
65     graph[0].push_back(make_pair(1, 2));
66     graph[1].push_back(make_pair(0, 2));
67     graph[1].push_back(make_pair(2, 3));
68     graph[2].push_back(make_pair(1, 3));
69     graph[2].push_back(make_pair(3, 1));
70     graph[3].push_back(make_pair(2, 1));
71     graph[3].push_back(make_pair(4, 6));
72     graph[4].push_back(make_pair(3, 6));
73
74     vector<Edge> MST_edges = MST(vertex, graph);
75     printMST(MST_edges);
76 }
PROBLEMS OUTPUT DEBUG CONSOLE PORTS
TERMINAL
janipassas@Janis-MacBook-Pro Homework % g++ -std=c++11 TestingMST.cpp -o TestingMST
janipassas@Janis-MacBook-Pro Homework % ./TestingMST
The edges in MST are given as:
0 - 1 : 2
1 - 2 : 3
2 - 3 : 1
3 - 4 : 6
```

Figure 2: Backend Output of MST using Prim's Algorithm

## 9.2 Challenges Faced

1. First challenge was learning and understanding backtracking algorithms.
2. Another challenge was changing the backtracking algorithm to be for a robot navigation concept instead of the in class example.
3. One challenge that was faced when implementing the algorithm was outputting one correct path when multiple correct paths existed.
4. Understanding Prim's algorithm, reading documentation and how to implement it in code.
5. Creating MST and introduction to graph problems in general was difficult.
6. Introducing obstacles and charging stations into the backtracking problem has not yet been added but will be the next steps.
7. Combining the MST and backtracking has not yet been done but are also next steps.

## 10 Action Items (November 10th)

### 10.1 Description

The team decided to create 4 action items that would be met by November 10th. The sections below illustrate the 4 target deliverables that were met by the team.

### 10.2 Action Item 1: Web Server UI

One of the main goals from the beginning of this project was the fact that the team wanted to not only have the output displayed on the terminal, but also have a comprehensible and easy-to-use UI to allow the user to fully use the backtracking algorithm used in the backend. A picture of the UI can be seen in Figure 6 of the appendix.

### 10.3 Action Item 2: Minimum Path Based on Weights

One of the big goals for this project was to incorporate weighted values for the path along the destination. This is applicable in the case of traffic along a path, or if its for navigation in a warehouse then a narrow passage that needs to be traversed carefully. This weighted input could be anything like fuel usage, time, money (tolls), or simply arbitrary. What the program does now is that it dynamically finds allPaths from the top left to the bottom right, and then it updates the object for pathTaken based on the accumulated weight. The resulting output is a grid showing 0s and 1s for the path taken (better for UI displaying), and then a print statement saying the total weight of this path. Overall, with this code, the minimum path based on the weights from the input maze is found and the path taken and its weight is then showed to the user.

### 10.4 Action Item 3: Flexibility in Movement, Maze Size, and Start/End Location

While testing the previous results, it was found that limiting the traversal to up and left was an issue in the scenario that a path was more complex and required zig zags or other more complex movement. This also became more apparent with the weighted paths since more non-linear paths are explored. To change this, movement up, left, right, and down are explored. This was done to find all the paths from the start and end. Some big roadblocks for this section was getting segmentation faults due to infinite loops in trying to dynamically get all the paths, and this was solved by setting nodes to 0 if they had been visited already. Once more complex movement was allowed, some code changes made it such that the input maze size can be a rectangle instead of a square which is nice for future testing and applications. Another change was to allow users to define a start and end location but requires them to pick a valid location (within the limits and not a 0). One limitation found was that a grid that is too large will not run properly, what happened was the computer will run out of memory space if the maze is larger than a 8x10. That being said, with this new code, minimum paths can be more complex, the maze size is more flexible, and the user can define start and end locations.

## 10.5 Action Item 4: Converting Backend Output in CSV File

One of the main roadblocks with the UI was the fact that it was extremely difficult to get data from the backend and have it displayed on the frontend. The myriad of function calls and use of third party services to generate the necessary JavaScript files from our C++ backend was painful. The team decided to take a step back and just focus on the basics of frontend development, which was HTML. An easy task in HTML is displaying a csv file, which is what the team did with our C++ backend. Now, the backend delivers a csv file that the frontend can easily decipher and display. Below in the appendix an example csv file can be seen in Figure 7.

## 11 Final Iteration (November 17th)

### 11.1 Description

The team decided to start the final presentation which can be seen here with this link: [https://northeastern-my.sharepoint.com/:p:/g/personal/malisetty\\_v\\_northeastern\\_edu/ESAWK0oHKV9GoUko4tIEBhYBB-e=3iP85N](https://northeastern-my.sharepoint.com/:p:/g/personal/malisetty_v_northeastern_edu/ESAWK0oHKV9GoUko4tIEBhYBB-e=3iP85N).

## 12 Appendix:

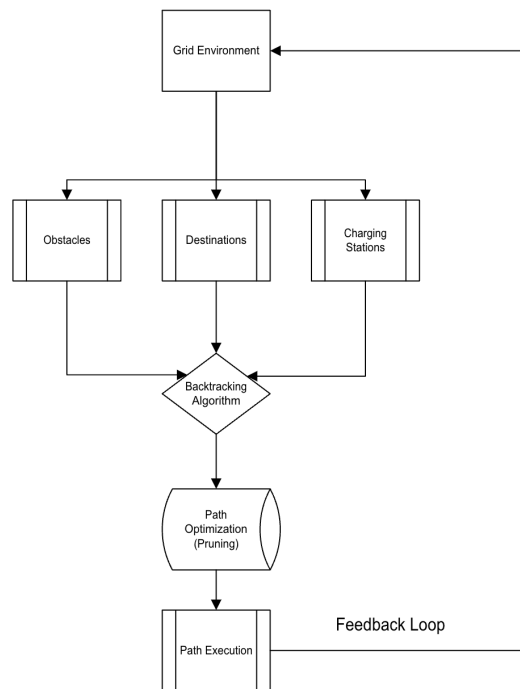


Figure 3: Algorithm Flowchart

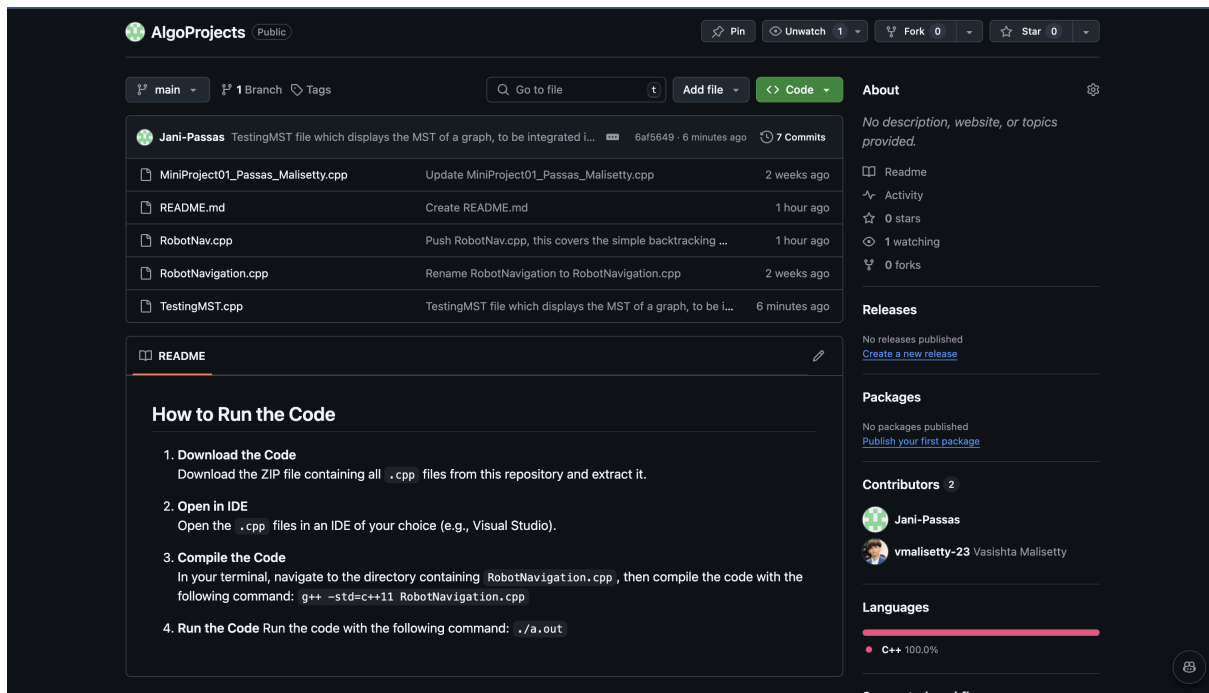


Figure 4: Screenshot of Github page with Project Document showing progress

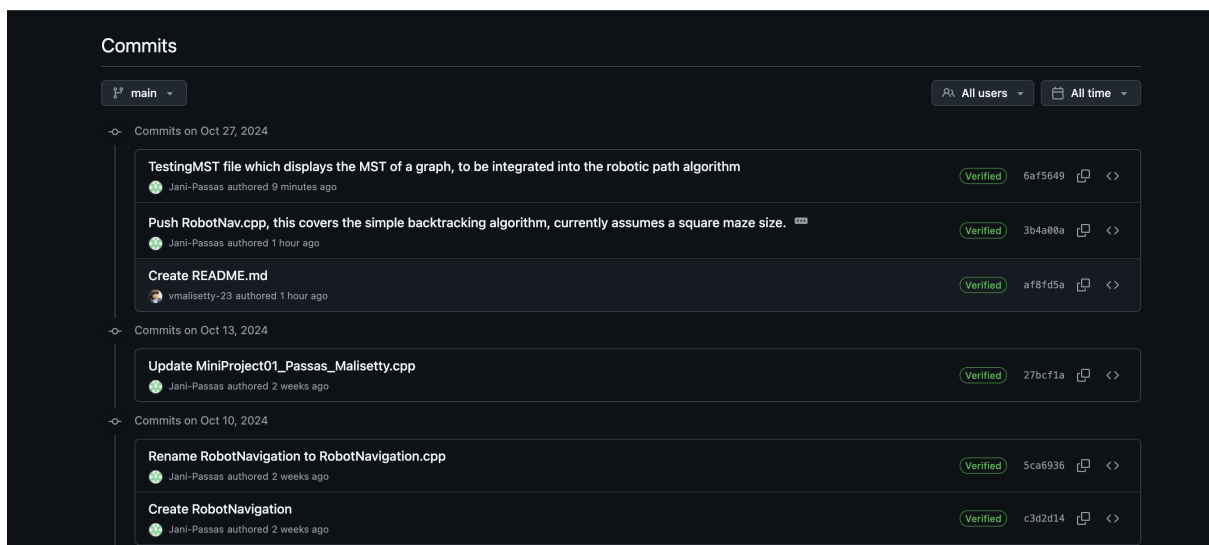


Figure 5: Commit History of Github showing finalized code before integration

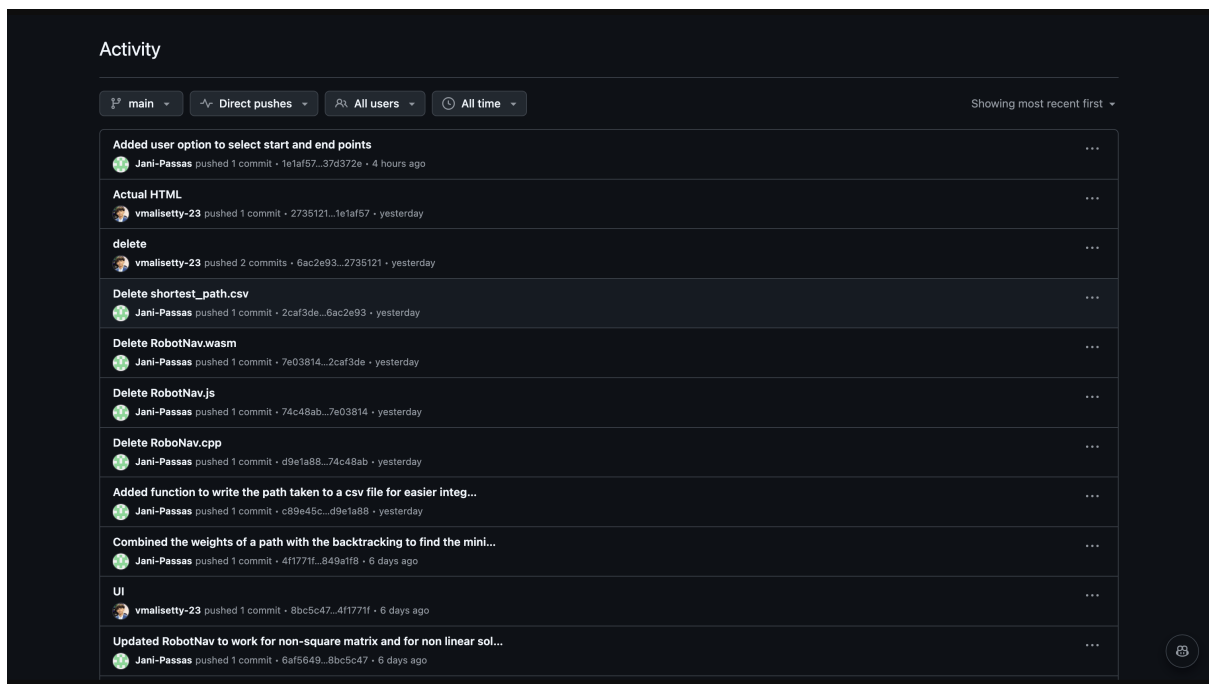


Figure 6: Commit History on 11/10 Iteration

```

TERMINAL zsh - Homework
janipassas@Janis-MacBook-Pro Homework % ./SampleRobotNavT
1 0 2 3 1 3 5 2 2 1
2 3 1 0 2 8 2 5 3 2
0 2 6 4 3 4 3 2 5 1
2 3 1 0 2 1 7 2 6 1
0 2 6 4 3 0 2 2 4 5
1 7 2 0 1 1 5 2 1 0

Boundaries for the input are 5 and 9
Please enter start coordinates (x y): 1 2
Please enter end coordinates (x y): 4 8
Minimum path weight: 29
Path grid given as:
0 0 1 1 1 0 0 0 0 0
0 0 1 0 1 0 0 0 0 0
0 0 0 0 1 1 1 1 0 0
0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 1 1
0 0 0 0 0 0 0 0 0 0

Data saved into CSV file and written to: Test1.csv
janipassas@Janis-MacBook-Pro Homework %

```

Figure 7: Output of More Advanced Path, User Defined Start/End, and Larger Maze



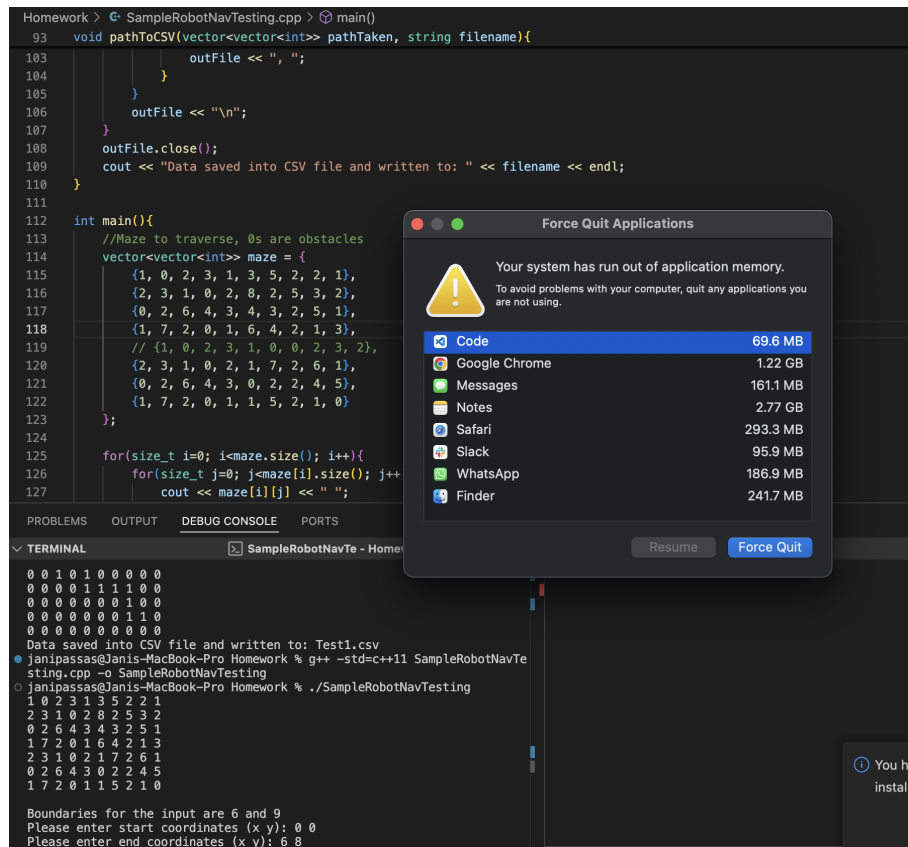


Figure 8: Error Message for if Maze Size is too Large

# Shortest Path Viewer

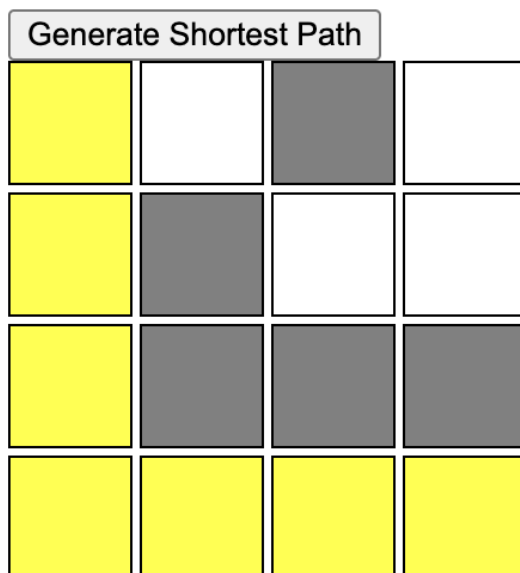


Figure 9: UI Version 1

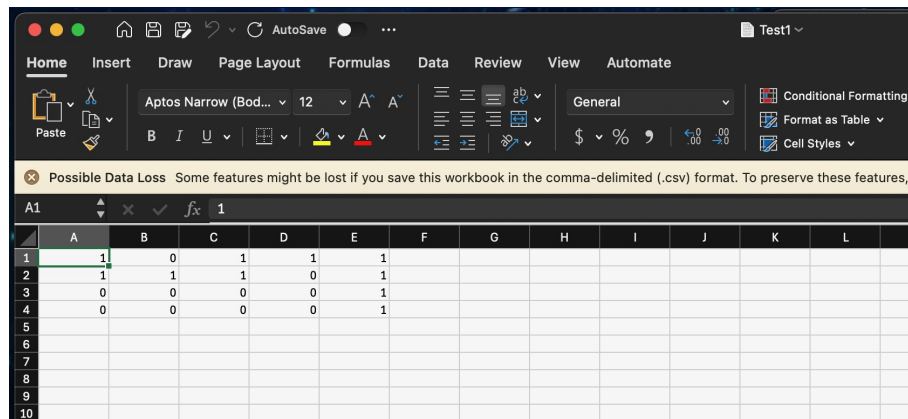


Figure 10: Output CSV file