

Final Report: Autonomous Delivery Robot Navigation

Team Name: JanVas

December 4, 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 as well as a responsive UI. The project's main objectives are:

- To allow delivery vehicle to most efficiently reach an endpoint.
- To enable robot to autonomously navigate through a maze.
- To develop a robust backtracking algorithm that can be applied to other fields.
- If time allows, expand into greedy algorithms and incorporate 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 back-end algorithm and output, a 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 & 9 (November 25 - December 4):** 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 (December 4th).

2.3 Team Roles

- **Vasishta Malisetty:** Responsible for front-end design and user interface development. This is due to the fact that Vasishta has previous experience in frontend development, specifically working with HTML and CSS. These skills have been further boosted through extracurricular involvements such as Forge and Generate Product Development Studio. Vasishta also possesses previous experience with Git and GitHub
- **Jani Passas:** Back-end developer, focusing on the terminal and back-end logic/algorithms. This is due to the fact that Jani has previous experience working with C++ code through coursework at Northeastern University as well as various extracurricular involvements such as NU Rover and Generate Product Development Studio. Jani also possesses previous experience with Git and GitHub

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++ 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++ (Live Server), HTML and CSS.
- **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, these include:

- 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++ 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

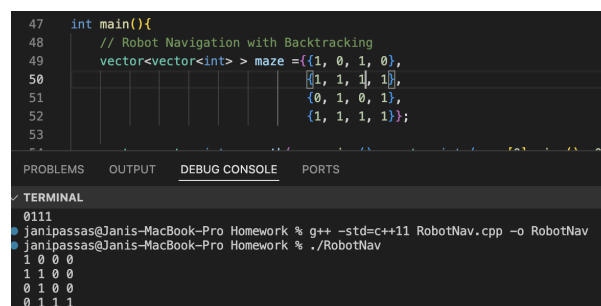
Step 4: Prune the brute force solutions until you get an optimal solution

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
54     ...
55 }
```

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 be 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 size 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 Presentation (November 17th)

11.1 Description

Our final presentation can be seen here with this link: https://northeastern-my.sharepoint.com/:p:/g/personal/malissetty_v_northeastern_edu/ESAWK0oHKV9GoUko4tIEBhYBB60duwWJsFm8M2G2xmdJXA?e=3iP85N.

12 Project Closure (December 4th)

12.1 Methodology

The team created a greedy and dynamic algorithm to navigate through a maze from a start to end point while charging along the way. There is an added complexity of fuel which is decremented each time a node is traversed, if the fuel reaches zero then the path is invalid. If a path is found, then a csv is generated which is used in a UI for the user. The data structures used for this project include structs, vectors, and queues. The two main approaches used were dynamic programming and greedy algorithm

12.2 Data Structures

Structs:

- Path
 - Data structure for the path followed, this is used for dynamic approach. Attributes include total weight, 2D vector showing path, and the fuel remaining
- Node
 - Used for each cell and in the priority queue for the greedy method. Attributes include the x, y coordinates, weight, and a boolean comparison for comparing weights with another node.

Vectors:

- 2D matrix of maze
- Used for path tracking
- Vector pair used for directions (up, down, left, and right)

Queues:

- Priority queue uses min-heap to always select smallest accumulated weight. Used in greedy method to always explore the local minimum path.

12.3 Approaches

Dynamic Programming:

- Recursively finds all paths
 - Divides into 4 sub-problems by moving in each direction
- Then picks final path with minimum total weight
 - If a tie in weight, then whichever has more remaining fuel is selected

Greedy Algorithm:

- Prim's MST and Dijkstra's inspired
 - Use priority queue/min-heap to move in the lowest weight direction
 - This code is more for traversing maze and start/end points rather than connecting a tree

Helper Functions:

- PrintPath (in terminal interface)
- PathToCSV (used for GUI)
- isValid (checks if a node is within the boundaries or an obstacle)
- hasAdjacentCharger (used in greedy method to charge if a neighboring cell is a charging station)

12.4 System Overview

- User Inputs
 - Maze currently hard coded but can be edited in main function
 - Start/end positions from user
 - Starting fuel and fuel capacity from user
- Dynamic or Greedy Approach (selected by user)
 - Both include charging stations and obstacles avoidance
- Output
 - Terminal (maze traversal, weight, and fuel info)
 - CSV (shows path and chargers)
 - UI (generated from the csv file)

12.5 Pseudocode

- `sValid(x, y, maze)`
 - Check boundary constraints and if cell is an obstacle
 - Returns true if valid, otherwise false
 - **isValid: $O(1)$**
- `HasAdjacentCharger(maze, x, y, rows, cols)`
 - Check possible moves (up, left, down, right)
 - If neighbors are within bounds and is a charger, return true
 - **hasAdjacentCharger: $O(1)$**
- `PrintPath(finalPath, maze)`
 - For each cell in matrix
 - if it is in path, print "1"
 - If it's a charger on path or neighbor, print "C"
 - If it's an unused charger, print "c"
 - Else, print "0"
 - **printPath: $O(n*m)$**
- `PathToCSV(pathTaken, filename)`
 - Open file with filename
 - Iterate through rows and write the row into the file (comma separated)
 - Close the file
 - **PathToCSV: $O(n*m)$**
- `FindAllPaths(maze, x, y, currentPath, allPaths, curWeight, endX, endY)`
 - If out of bounds, return
 - If destination, add path to allPaths with weight
 - If cell already visited, return
 - For each direction if newCell isValid, recursively call findAllPaths to explore deeper into path until destination reached or hit dead end
- **Time complexity: $O(4^{(n*m)})$**
- `FindMinPath(allPaths)`
 - Initialize minPath
 - For each path in allPaths, if path weight is less than minPath, then minPath = current path
 - Return minPath
 - **Time complexity: $O(n)$**

- findGreedyPath(maze, startX, startY, endX, endY, greedyPath, fuel, maxFuel, fuelExhaustion)
 - Initialize visited array, parent array, usedChargers array, and priority queue
 - While priority queue not empty
 - * Dequeue node with smallest weight
 - If destination, trace back to start using parent array
 - * Mark path in greedyPath and return total weight
 - for each direction, move and check isValid and not visited.
 - * Update weight, visit status, and parent information.
 - * Enqueue new cell into priority queue
 - If destination not reached, return -1
 - **Time complexity: $O((n*m) \log(n*m))$**
- LoadCSVFile()
 - Fetch CSV file
 - **Time Complexity: $O(n)$**
- RenderGrid()
 - Use a nested for loop to iterate through every element within the CSV file
 - If grid == 1, add block to path
 - Else, do not add block to the path
 - **Time Complexity: $O(n^2)$**

12.6 Results

The results of our program is a maze traversal code where the user decides the start and end coordinates, the type of algorithm run (greedy or dynamic), and fuel parameters. The terminal shows the final path weight, which can represent time, the final path grid, as well as the remaining fuel. As added features, we have differentiated the chargers used and the unused chargers which is also displayed in the UI. The user is able to change the maze weights and obstacles, but future steps involve making this an option to edit in a fully responsive UI. Future steps also include adding multiple way points instead of just one end point, but this would involve changing the dijkstra's approach to something closer to Prim's MST.

```

vmalisetty_23@Vasishtas-MacBook-Pro AlgoProjects % ./a.out
1 0 2 -1 7 6
2 3 1 0 2 8
0 2 5 4 3 4
1 7 0 0 1 6
2 -1 1 0 2 1
0 2 6 4 3 0

Boundaries for the input are 5 and 5
Please enter start coordinates (x y): 0 0
Please enter end coordinates (x y): 4 4
Please enter starting fuel: 20
Please enter max fuel capacity: 30
Please enter 1 for dynamic approach and 2 for greedy approach: 1
Minimum path weight: 25
Path grid given as:
1 0 1 C 1 0
1 1 1 0 1 0
0 0 0 0 1 0
0 0 0 0 1 0
0 c 0 0 1 0
0 0 0 0 0 0

Remaining fuel: 15
Data saved into CSV file and written to: output.csv
vmalisetty_23@Vasishtas-MacBook-Pro AlgoProjects %

```

Figure 3: Dynamic Programming Output

Note that the dynamic grid is required to be smaller due to the large memory space used in this approach. But a global optimal solution is guaranteed.

Backtracking UI

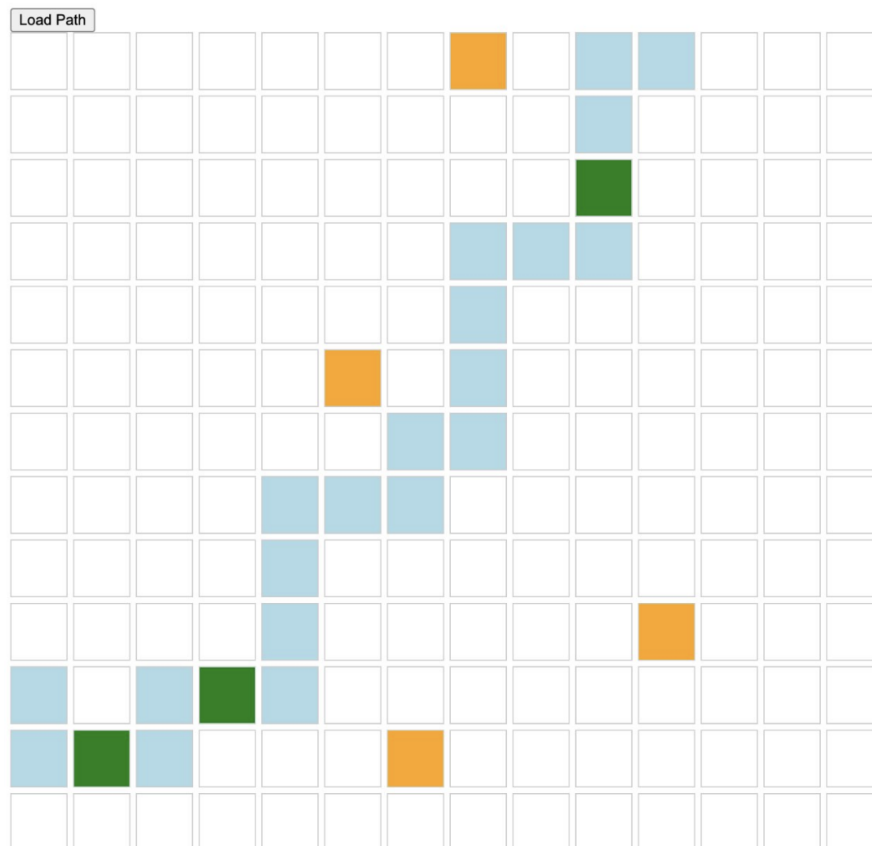


Figure 6: Final GUI

The UI uses baby blue blocks to depict the shortest path, green blocks to depict used chargers, and orange blocks to depict unused chargers

```

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

Boundaries for the input are 13 and 13
Please enter start coordinates (x y): 1 1
Please enter end coordinates (x y): 12 12
Please enter starting fuel: 15
Please enter max fuel capacity: 30
Please enter 1 for dynamic approach and 2 for greedy approach: 2
Minimum path weight: 49
Path traveled was:
0 0 1 1 1 1 1 1 C 0 0 0 0 0 0
0 1 1 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 1 0 c 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 c 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 1 1 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 0 0 0 0
0 0 0 0 0 0 0 0 0 1 C 1 0 0
0 0 0 c 0 0 0 0 0 0 0 1 1 0
0 c 0 0 0 0 c 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0
Remaining fuel: 19
Data saved into CSV file and written to: output.csv
janipassas@Janis-MacBook-Pro Homework %

```

Figure 4: Greedy Algorithm Output

The greedy algorithm works much faster and can be scaled up much easier, however this cannot guarantee a global optimal solution since decisions are made locally.

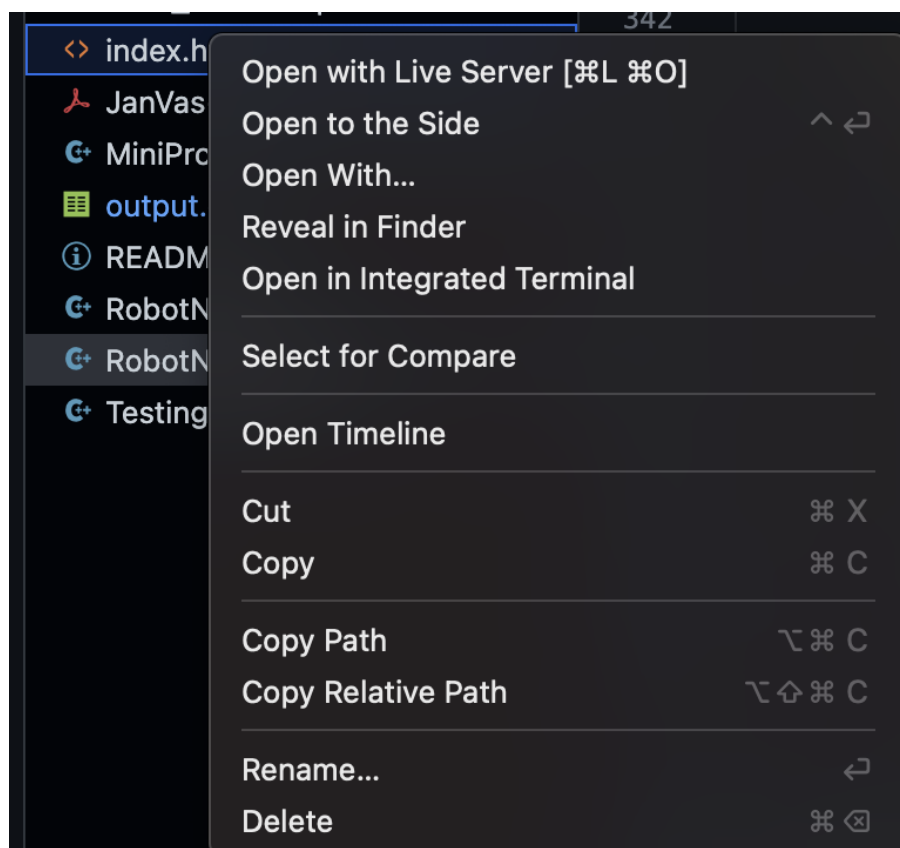


Figure 7: Live Server

The Live Server extension is used to host our web browser server that runs the HTML and CSS code

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	0	0	1	1	1	1	1	-2	0	0	0	0	0	0	0
2	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	1	0	-1	0	0	0	0	0
4	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
6	0	0	0	0	0	-1	0	1	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	1	-2	1	0	0	0
11	0	0	0	-1	0	0	0	0	0	0	0	1	1	0	0
12	0	-1	0	0	0	0	-1	0	0	0	0	0	1	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15															
16															

Figure 5: Output CSV Generated by Back-end, Used by Front-end

```

6      <title>Backtracking UI</title>
7      <style>
8          .grid-container {
9              display: grid;
10             grid-template-columns: repeat(6, 1fr);
11             grid-template-rows: repeat(6, 1fr);
12             gap: 5px;
13             width: 300px;
14             height: 300px;
15         }
16         .grid-item {
17             width: 50px;
18             height: 50px;
19             display: flex;
20             align-items: center;
21             justify-content: center;
22             border: 1px solid #ccc;
23             text-align: center;
24             font-size: 12px;
25         }
26         .blue { background-color: lightblue; }
27         .green { background-color: green; }
28         .grey { background-color: grey; }
29         .obstacle { background-color: white; }
30         .charger { background-color: green; }
31         .path { background-color: green; }
32         .unused { background-color: white; }
33     </style>
34 </head>

```

Figure 8: HTML and CSS Code

This is a code snippet of the CSS code that was used to create the appropriate sizing for our grid

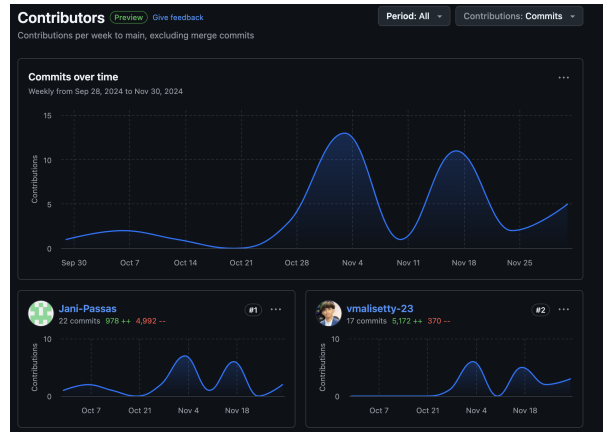


Figure 9: Git Commit History

This is the Git commit history between Vasishta and Jani. This shows that the team started the project early and often in order to make the deadlines

12.7 Discussion

Limitations:

While our code works as intended, it has limitations that need to be addressed for it to be useful in real-world scenarios. As of right now, everything is based on an ideal environment where all inputs and conditions are ideal (no variance). This lack of realism makes it impossible to adapt to changes such as dynamic obstacles or shifting environmental factors. Another issue is the limited memory space, which makes it hard to scale up for larger applications. Because of this, dynamic programming can't be used effectively to find the best global solution.

Applications:

The potential applications include planning energy-efficient travel, where a vehicle finds the best route from a start point to a destination while factoring in necessary charging stops. This is especially relevant for optimizing electric vehicle performance. Another real-world use case could be in robotics, like Room-bas or delivery bots, where the goal is to navigate from one point to another while avoiding obstacles. Whether its in a warehouse automation or drone deliveries or something entirely different, this navigation algorithm practical implications. By addressing these current limitations, more future possibilities could be explored which could make the system have much more impact.

12.8 Conclusion

- Overall, our project successfully completed our requirements of traversing a maze efficiently, charging, and displaying the outputs in a user friendly manner. While there were many iterations of code, and a few issues came up with integration and within the back-end algorithms, we worked through and found solutions to get the end product. This was a valuable project in exploring greedy and dynamic programming by expanding beyond what was learned in class, as well as creating helper functions, and integrating a priority queue/min-heap into a larger algorithm. We are happy with our final product and it is successful based on the following criteria.
 - Successfully connected front-end & back-end code
 - Both Dynamic Programming and Greedy Algorithms work as expected
 - UI is standalone and works properly
- Future steps
 - Making a fully responsive UI would be a good next step. This would allow the user to set obstacles, weights, and waypoints in the front end without having to interact with the back end terminal at all.
 - Multiple waypoints with a Prim's MST algorithm. While this was not in our problem statement and was not expected to be in our project, this could be another step for a project like this to show how different algorithms have different use cases and this also dives further into graph theory than we currently have.

12.9 References

1. Amazon Science. "The Science Behind Grouping Amazon Package Deliveries." *Amazon Science*, <https://www.amazon.science/latest-news/the-science-behind-grouping-amazon-package-deliveries>. Accessed 4 Dec. 2024.
2. ScienceDirect. "This Method Refers to the Current Research Presented in Ref." *ScienceDirect*, <https://www.sciencedirect.com/science/article/pii/S0142061524001522>. Accessed 4 Dec. 2024.
3. "Prim's Minimum Spanning Tree (MST) – Greedy Algorithm." *GeeksforGeeks*, <https://www.geeksforgeeks.org/prim-minimum-spanning-tree-mst-greedy-algo-5/>. Accessed 4 Dec. 2024.
4. "Rat in a Maze." *GeeksforGeeks*, <https://www.geeksforgeeks.org/rat-in-a-maze/>. Accessed 4 Dec. 2024.
5. "Introduction to Dijkstra's Shortest Path Algorithm." *GeeksforGeeks*, <https://www.geeksforgeeks.org/introduction-to-dijkstras-shortest-path-algorithm/>. Accessed 4 Dec. 2024.

6. "How to Create Input Field That Accepts CSV File in HTML." *GeeksforGeeks*, <https://www.geeksforgeeks.org/how-to-create-input-field-that-accept-csv-file-in-html/>. Accessed 4 Dec. 2024.
7. "How to Get a 5 by 5 Grid in CSS." *Stack Overflow*, <https://stackoverflow.com/questions/46780038/how-to-get-a-5-by-5-grid-in-css>. Accessed 4 Dec. 2024.

13 Appendix:

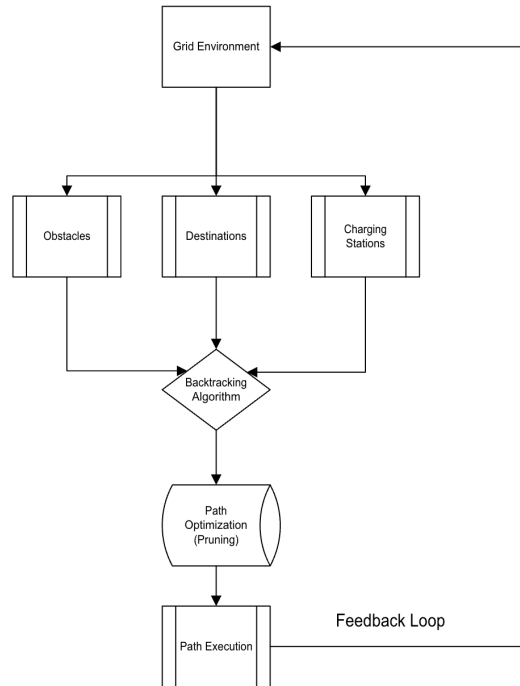


Figure 10: Algorithm Flowchart

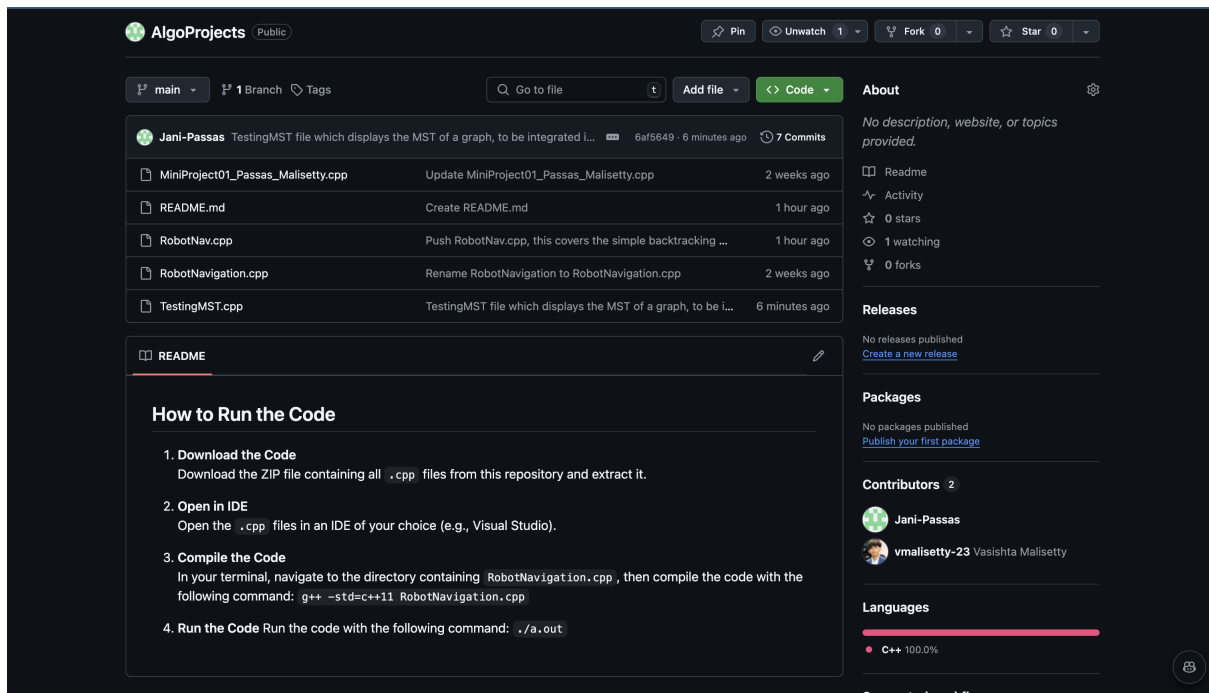


Figure 11: Screenshot of Github page with Project Document showing progress

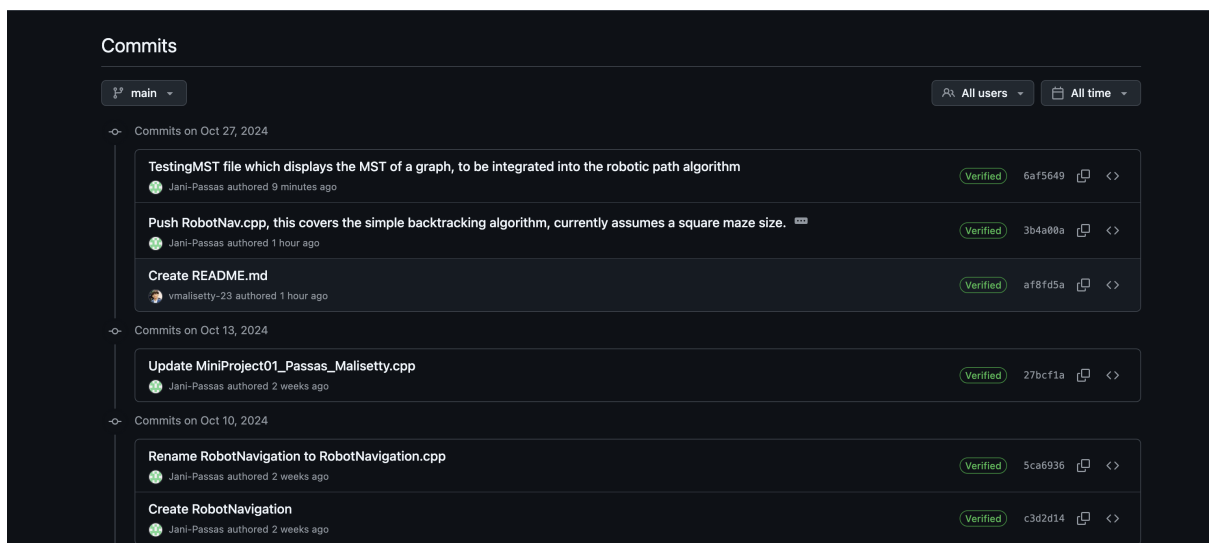


Figure 12: Commit History of Github showing finalized code before integration

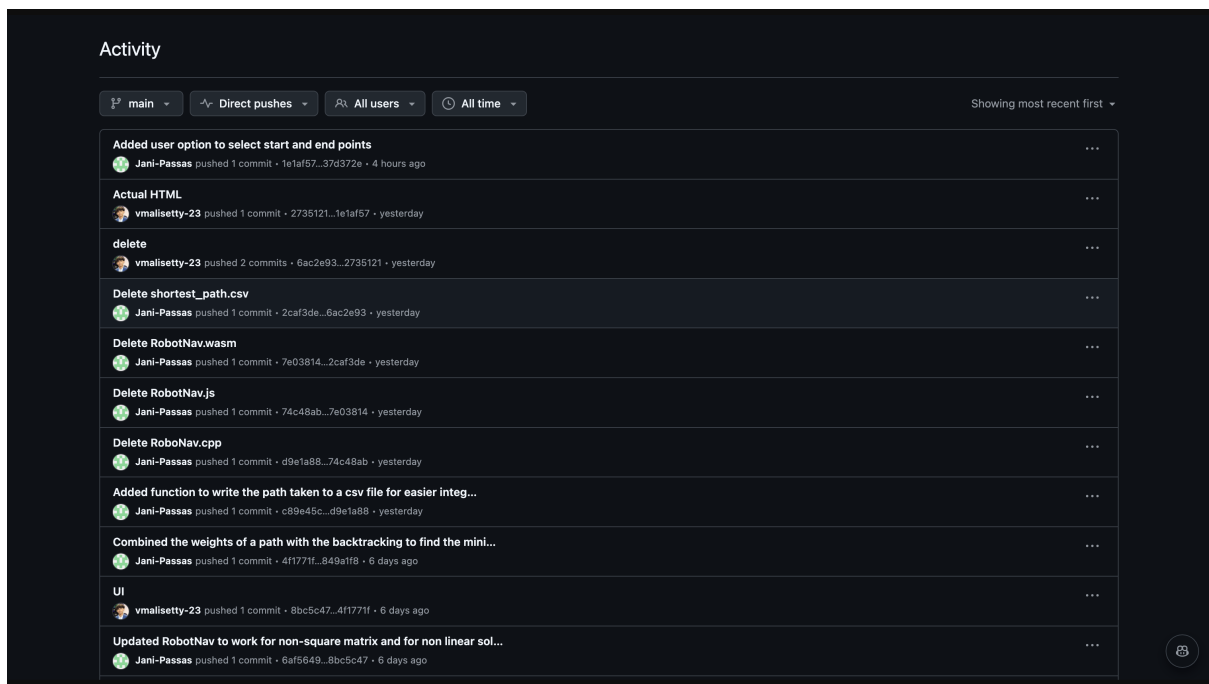


Figure 13: Commit History on 11/10 Iteration

```

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 1 0 0
0 0 0 0 0 0 0 1 1 0
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 14: Output of More Advanced Path, User Defined Start/End, and Larger Maze

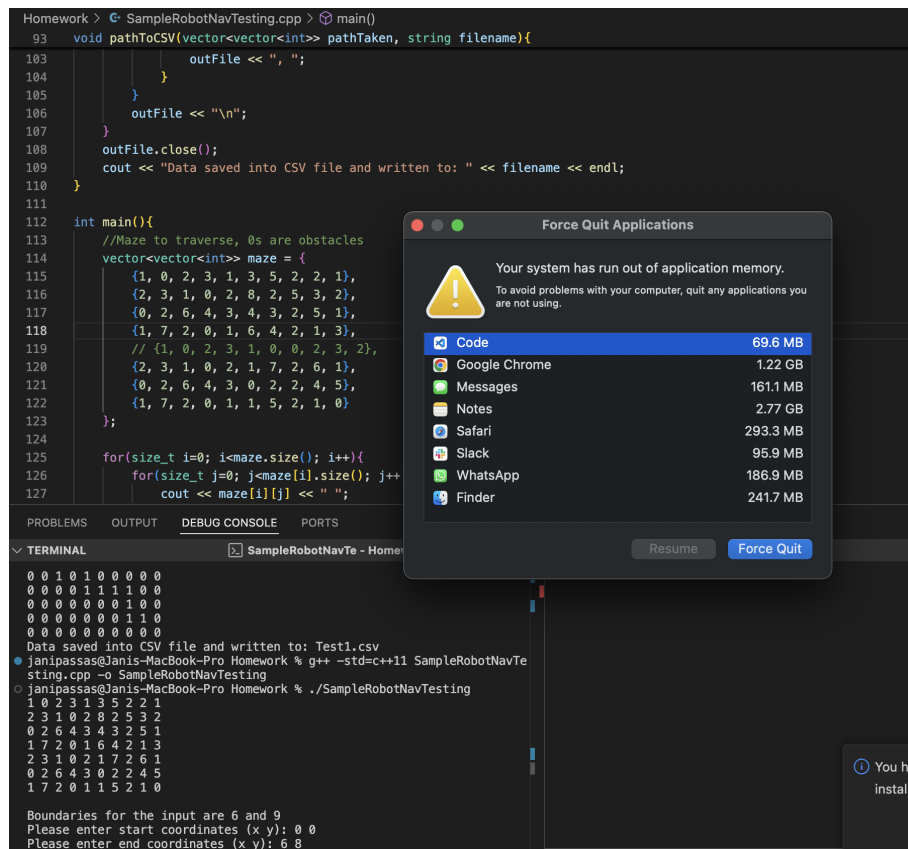


Figure 15: Error Message for if Maze Size is too Large