Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavre

A Project Report
on
“Maze Generation and Pathfinding using Graph Algorithms : A DSA Mini Project ”

[Code No:COMP 202]

(For partial fulfillment of Year II /Semester I in Computer Engineering)

Submitted by

Aayusha Acharya (01)

Sarbesh Paudel (35)

Aadim Sapkota (44)

Submitted to

Sagar Acharya

Department of Computer Science and Engineering

Submission Date: February 22, 2026

Bona fide Certificate

This project work on

“Maze Generation and Pathfinding using Graph Algorithms : A DSA Mini Project”

is the bona fide work of

“

Aayusha Acharya (1)

Sarbesh Paudel (35)

Aadim Sapkota (44)

”

who carried out the project work for the partial fulfillment of Year II /Semester I in Computer Engineering .

Acknowledgement

We would like to express our sincere gratitude to the Department of Computer Science and Engineering for providing us with this remarkable platform and resources for carrying out this project as a part of our coursework. This project has allowed us to broaden our perspective, strengthen our theoretical understanding in DSA by applying it practically through visualization.

Our utmost appreciation goes to our lecturer, Er. Sagar Acharya, for his guidance, encouragement and right directions. His lectures, discussions, and academic insights laid the strong foundation that enabled us to complete this project successfully. Furthermore, we acknowledge and admire the support, motivation we received from our family members and our friends.

Aayusha Acharya (1)

Sarbesh Paudel (35)

Aadim Sapkota (44)

Abstract

Graph algorithms are an important concept in computer engineering, often used in games, robotics, and navigation systems for pathfinding purposes. In this project, we have built a graphical tool that allows users to visualize a path in a generated maze. The tool was developed using C++ and Qt. Its primary focus is on static modes of mazes. In static mode, users can generate mazes and run algorithms on them. Users can select the start and end points using clicks, and the pathfinding algorithm can be selected from the menu on the side. The system includes common algorithms such as Breadth-First Search and Depth-First Search for pathfinding. The system also includes Kruskal's algorithm for maze generation. All of these are common graph algorithms within the premises of COMP 202.

*Keywords**:*** Pathfinding, Qt, Maze Generation, BFS, DFS, Kruskal's Algorithm

Table of Contents

List of Figures

Acronyms/Abbreviations

The list of all abbreviations used in the documentation is included below:

BFS- Breadth First Search

DFS- Depth First Search

CLI- Command Line Interface

GUI- Graphical User Interface

GPU- Graphics Processing Unit

Chapter 1 Introduction

This project revolves around a graphical application developed using C++ and Qt framework. The main goal of this project is to visualize the execution of different pathfinding and maze generation algorithms. It also serves as an educational tool that helps users to understand the vast concepts of data structures and algorithms.

Pathfinding algorithms are commonly used for generating GPS and Route Navigation systems by optimizing delivery routes, which ultimately saves travel time, reduces fuel consumption in transportation.

1.1 Background

A maze is a travel puzzle with complex branching passages where the explorer must find a route. The mission of this travel puzzle is to find the fastest route, or perhaps the only route to reach the destination. The destination location can be known or unknown. Pathfinding is the study of figuring out how to get from source to destination. Maze generation typically uses grid-based and graph-based algorithms. Pathfinding algorithms then find routes through it.

1.2 Objectives

The primary goal of this semester project includes:

- Allow users to generate maze in a grid using algorithms such as Kruskal's.
- A user interface so that users can select algorithms viz. BFS, DFS for pathfinding, set parameters and control the simulation (pause/play).

1.3 Motivation and Significance

The idea for this project also came from our course curriculum COMP 202 , where we studied graph algorithms under Data Structures and Algorithms, and wanted to bring those concepts to life through a hands-on visual application.These algorithms are also widely used in many fields, including robotics, game development, AI, and navigation systems, yet they often remain abstract and difficult to grasp without a clear visual representation.

The project allows users to test different algorithms and understand how they solve the maze. Besides helping people learn, it can also be used to test how pathfinding works in simple, generated setups. This can be useful for building more advanced systems in the future, incorporating more algorithms and even reinforcement learning agents.

Chapter 2 Literature Review

2.1 Comparative Analysis of Maze Generation Algorithms

Mane et al. (2024) provide an evaluation of several maze generation methods. They implement and compare Prim's, Kruskal's, recursive Depth-First Search (DFS) and Wilson's algorithms. Performance is measured by generation time. The maze generating algorithms are scored based on the performance of the agents. The algorithms' performance determines the most effective algorithms.

2.2 Comparative Pathfinding in a Maze Game

Permana et al. (2018) compare three core graph-search algorithms: A*, Dijkstra's, and Breadth-First Search (BFS), within a custom Maze Runner game. They embed each algorithm in the game's NPC AI and measure metrics like computation time, path length, and steps taken as players block the NPC's route. The study finds that A* typically finds the shortest path most quickly, whereas BFS and Dijkstra vary in cost depending on maze complexity.

2.3 Parallel Maze Generation and Solving

The research by Muthuselvi et al. (2016) implements a multi-core C++ solution for maze generation and solving using Kruskal's algorithm. The proposed system partitions the task in a balanced way so that each core has the same amount of job to do. The study reveals that the proposed system offers high performance and CPU utilization. It has been observed that the computation time taken by parallelized maze generator and solver is significantly less than sequential maze generator and solver.

Chapter 3 Design and Implementation

It includes the explanation of the sequential procedure that was performed during the project work. Each stage was carefully planned to ensure the system was functional.

3.1 System Requirement Specifications

This section specifies the software and hardware requirements of the developed system.

3.1.1 Software Specifications

- **Programming Languages** : C++
- **Operating System** : Windows, macOS or Linux.
- **Framework Used** : Qt
- **Build Automation Tool** : qmake
- **Version Control System** : Git
- **Application for execution** : Qt Creator 17.0 onwards

3.1.2 Hardware Specifications

- **Processor**: Intel i5/ Ryzen 5(for multithreading)
- **RAM**: Minimum 8GB (16GB recommended)
- **Storage**: SSD (256GB or more)
- **GPU**: NVIDIA GPU (optional)

3.2 Implemented Algorithms

We have implemented the following two categories of algorithms:

3.2.1 Algorithms used for Maze Generation

• **Kruskal's Algorithm**: The grid starts filled with walls. Each cell is treated as a separate set in a disjoint-set data structure. Walls between adjacent cells are collected and shuffled. The algorithm iterates through these walls, and for each wall, it checks if the two cells it separates belong to different sets. If so, it removes the wall, merges the sets, and adds both cells to the maze. This continues until all cells are connected without cycles. The result is a fully connected maze with no loops, optimized for randomness and uniformity.

3.2.2 Algorithms used for Pathfinding

• **Breadth-First Search(BFS)**: BFS explores nodes in a breadth-first manner using a queue, starting from the start node. Nodes are dequeued, processed, and their unvisited, non-obstacle neighbors are enqueued and marked visited. Parent mapping tracks the path. The search stops upon reaching the goal or exhausting nodes. After completion, it reconstructs the shortest path by backtracking parents. Visualization signals track visited nodes, next nodes, and the final path. Cancellation and suspension checks are integrated. Node states are reset after execution.

• **Depth-First Search (DFS)** : DFS explores graph depth-first using a stack. It starts from the start node, pushing neighbors onto the stack and marking them visited when added. Nodes are popped from the stack, processed, and their neighbors are explored recursively until the goal is found or no nodes remain. The parent map tracks the path for reconstruction after the goal is reached. DFS does not guarantee the shortest path.

3.3 GUI Overview

The GUI was created with Qt Creator's in-built Design Mode, structured into three primary tabs:

1. **Simulation Tab**
    A. Select pathfinding algorithm
    B. Set start and goal nodes

Figure 3.1: Select Interaction Dropdown

Figure 3.2: Select Algorithm Dropdown

2. Visualization Tab

A. Adjust horizontal and vertical node count

B. Set Marker Size

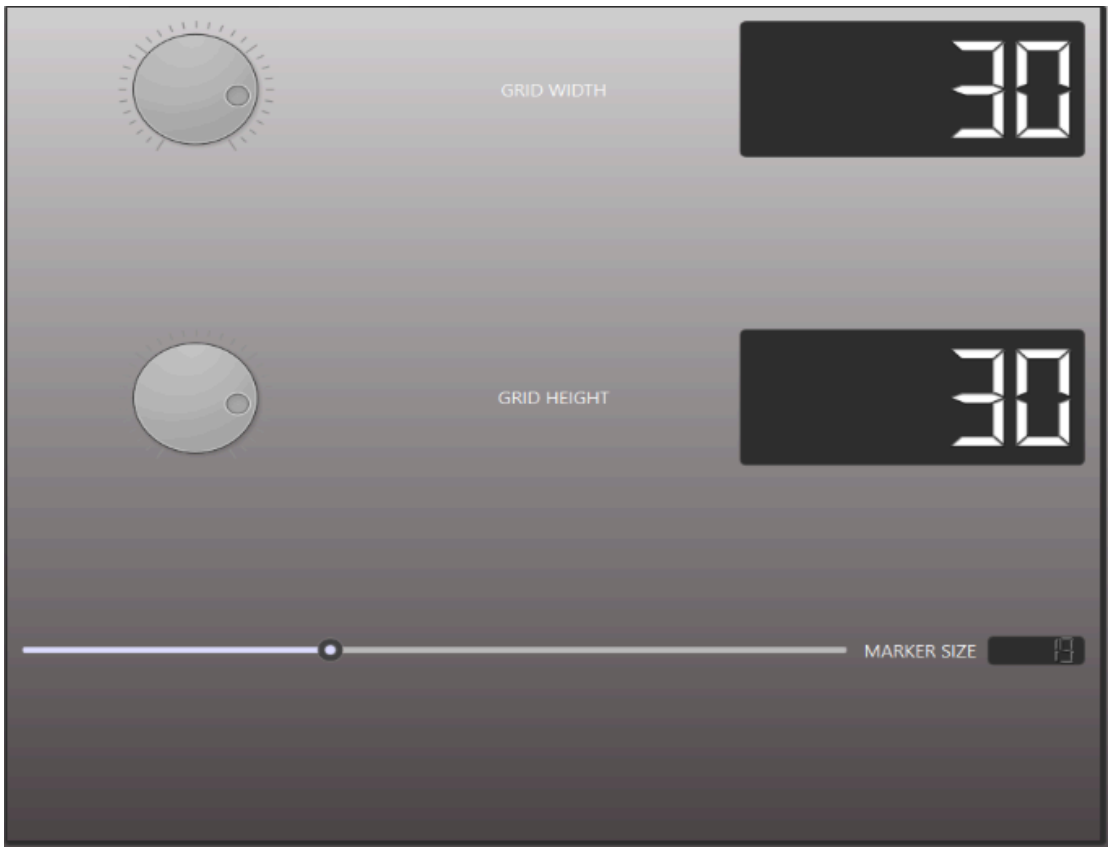

Figure 3.3: Visualization Tab

3.4 System Diagrams

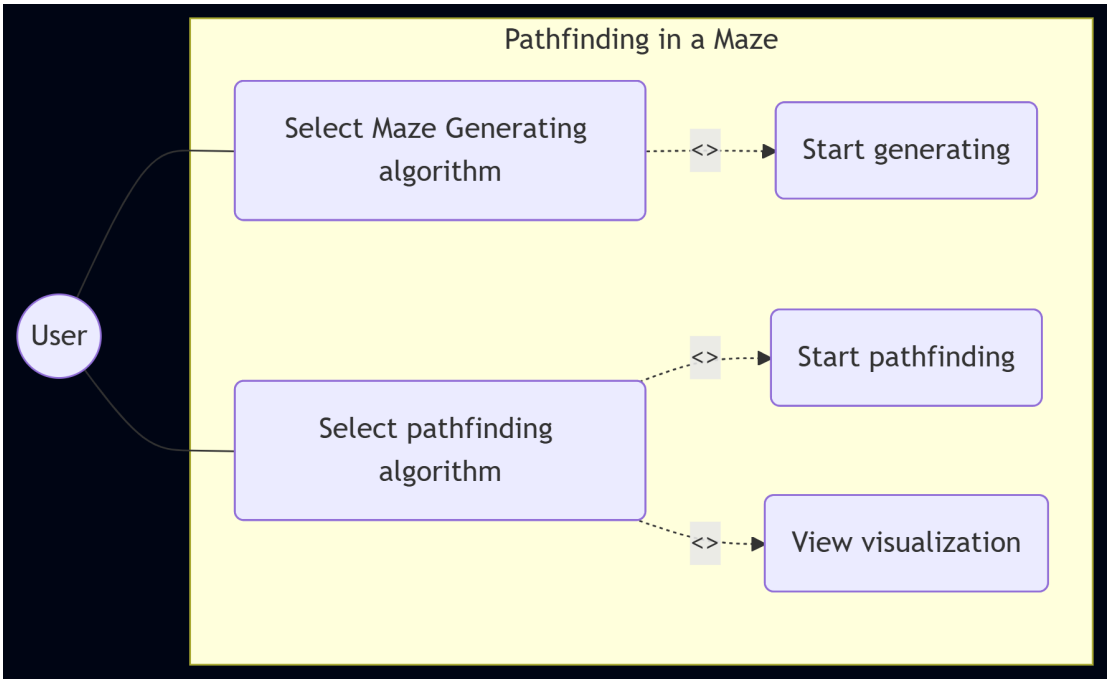

Figure 3.4: Use-case Diagram

A use case diagram is a visual representation of how a user interacts with a system to achieve a specific goal. Here, the "User" is the actor who interacts with the system. The user has two main use cases: selecting a maze generation algorithm, and selecting a pathfinding algorithm. The "Select Maze Generating algorithm" use case includes the option to start generating the maze. The "Select pathfinding algorithm" use case includes starting the pathfinding process and viewing the visualization.

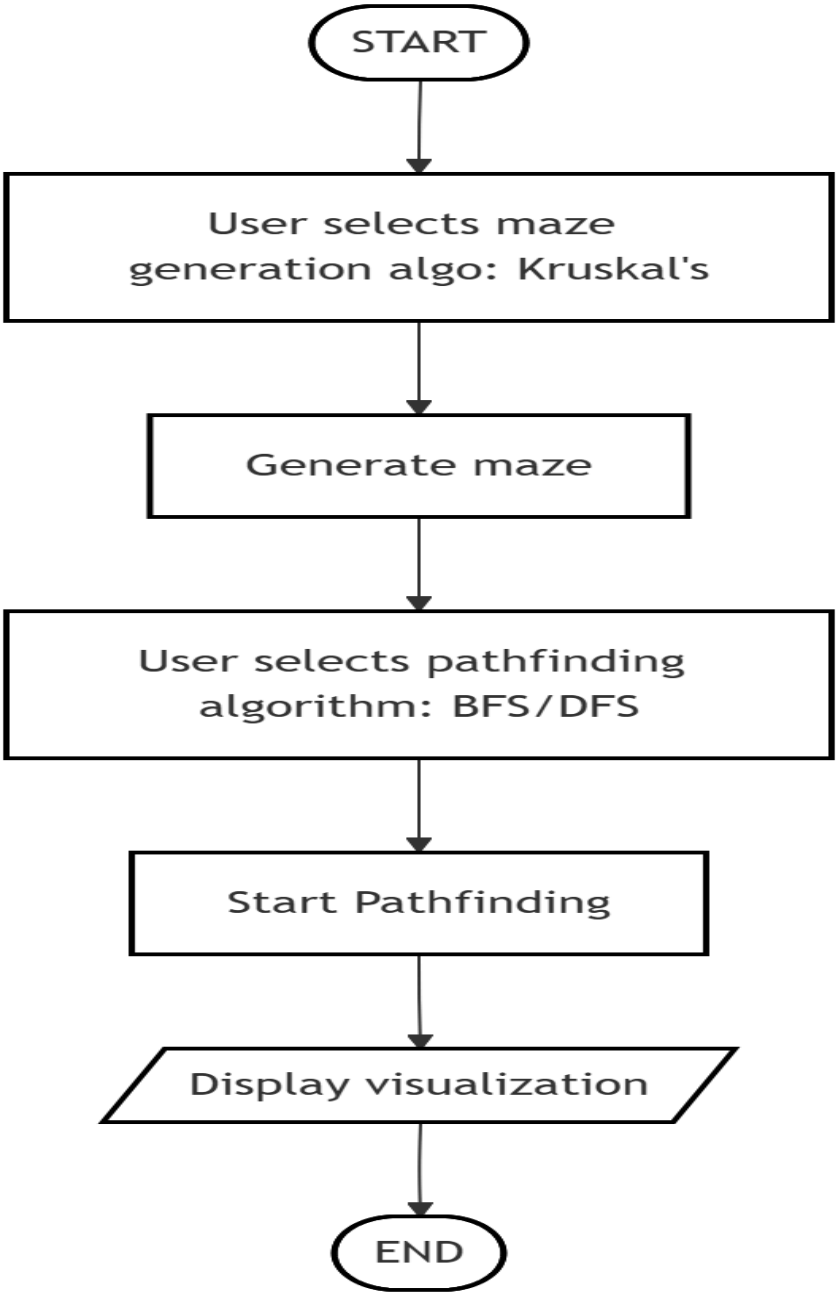

Figure 3.5: Workflow diagram

A workflow diagram shows the sequential steps of a process. This diagram outlines the process for a Maze Generation and Pathfinding System.

Chapter 4 Discussion on the achievements

We have implemented several critical features enhancing both functionality and user experience. These include interactive simulation controls for visualization such as pause/play features for both DFS and BFS algorithms on identical mazes. These additions collectively improve usability and extensibility of the system.

4.1 Features

We have successfully implemented key features that makes our project user-friendly:

- **Simulation Controls**: Our application enables users to pause, play the animations for both maze generation and pathfinding.
- **Multithreading**: Enables real-time simulation and visualization by dividing execution into independent threads.

4.2 Challenges

We faced a lot of challenges during the developmental phase. One of the main difficulties was to ensure that the GUI remained responsive and didn't get frozen during the visualization of the algorithms. The project required significant testing and debugging. Integrating the algorithm logic with the Qt framework was one of the key challenges, as making the visual updates sync with the algorithm's execution required careful handling. Overall, balancing the technical complexity of the algorithms with a clean and functional visual representation was the biggest challenge throughout the project.

Chapter 5 Conclusion and Future Works

After completing this project, all project deliverables were functionally implemented and tested. We successfully implemented pathfinding algorithms to find the route and avoid obstacles on a maze. The real highlight, however, was to create a graphical interface that demonstrates the entire process, making the overall setup of maze generation and pathfinding easier to understand.

The user interface we developed, also makes the entire experience seamless, allowing anyone to easily choose an algorithm.

5.1 Limitations

While our project covers a lot as of now, there's still room to grow. Right now, our biggest areas of improvement are in the depth of our algorithm analysis and the range of pathfinding techniques we have implemented.

- Right now, paths are traced as cardinal movements(up/down/left/right). The feature of diagonal movement would make it more realistic.
- Although the current algorithm techniques work for generated mazes, the application still lacks dynamicity while the simulation is running. The algorithm doesn't adjust to the changes made in the grid during runtime.

5.2 Future Enhancement

- Introduce bidirectional search algorithms that run simultaneously from the start and goal nodes to reduce search time and improve efficiency, especially in large mazes.
- Include more algorithms for maze generation and pathfinding, such as A*, Dijkstra, Prim's, Wilson's, etc.

- Implement a functionality that enables users to add or remove obstacles in real time within the maze.
- Integrate machine learning models trained on various maze structures (e.g., sparse, dense, looping) to adapt pathfinding strategies dynamically.
- Enhance the system to handle dynamic mazes where obstacles or goals change over time, requiring the agent to replan paths efficiently.
- Implement a " **Play Yourself** " mode, where users can actively participate and control the system's behaviour, to make the system interactive and user - centric.
- Include a performance evaluation module that generates a comparative table of different pathfinding algorithms based on the key performance metrics such as execution time, number of nodes visited, etc.

References

Karur, K., Sharma, N., Dharmatti, C., & Siegel, J. E. (2021). *A Survey of Path Planning Algorithms for Mobile Robots*. *Vehicles, 3*(3), 448–468.

Singh, R., Ren, J., & Lin, X. (2023). *A Review of Deep Reinforcement Learning Algorithms for Mobile Robot Path Planning*. *Vehicles, 5*(4), 1423–1451.

Permana, S. D. H., Bintoro, K. B. Y., Arifitama, B., & Syahputra, A. (2018). *Comparative analysis of pathfinding algorithms A*, Dijkstra, and BFS on Maze Runner game*. *International Journal of Information System and Technology*, 1(2), 1–6.

Mane, D., Harne, R., Pol, T., Asthagi, R., Shine, S., & Zope, B. (2024). *An extensive comparative analysis on different maze generation algorithms*. *International Journal of Intelligent Systems and Applications in Engineering*, 12(2s).

Muthuselvi, R., Sindhuja, A., & Sridevi, P. K. D. (2016). Parallelization of maze generation and solving in multicore using OpenMP. *International Journal of Innovative Research in Science, Engineering and Technology*, 5(Special Issue 11), 7–12.

APPENDIX

A Sample Outputs

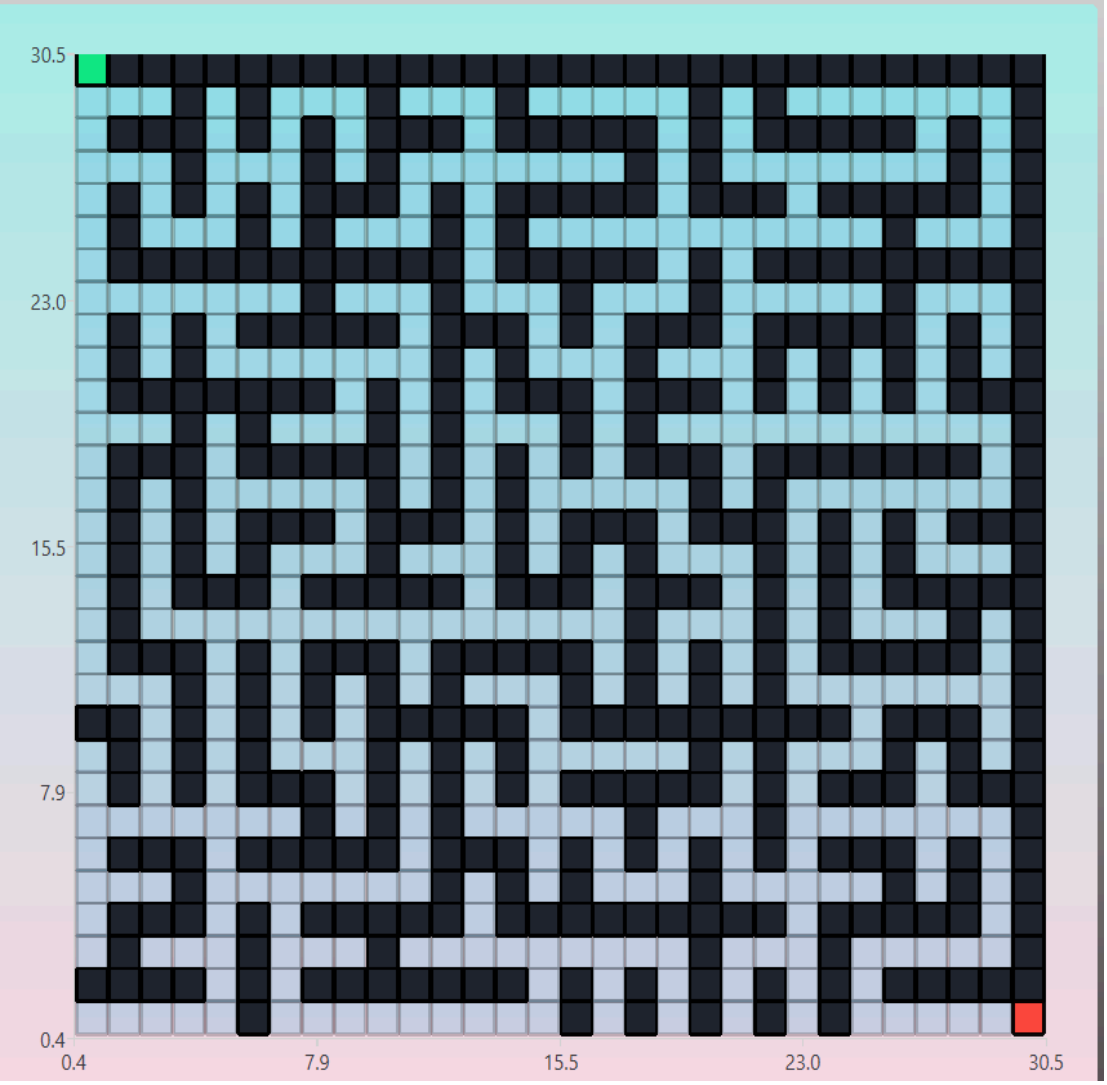

Figure A.1: Generated Maze

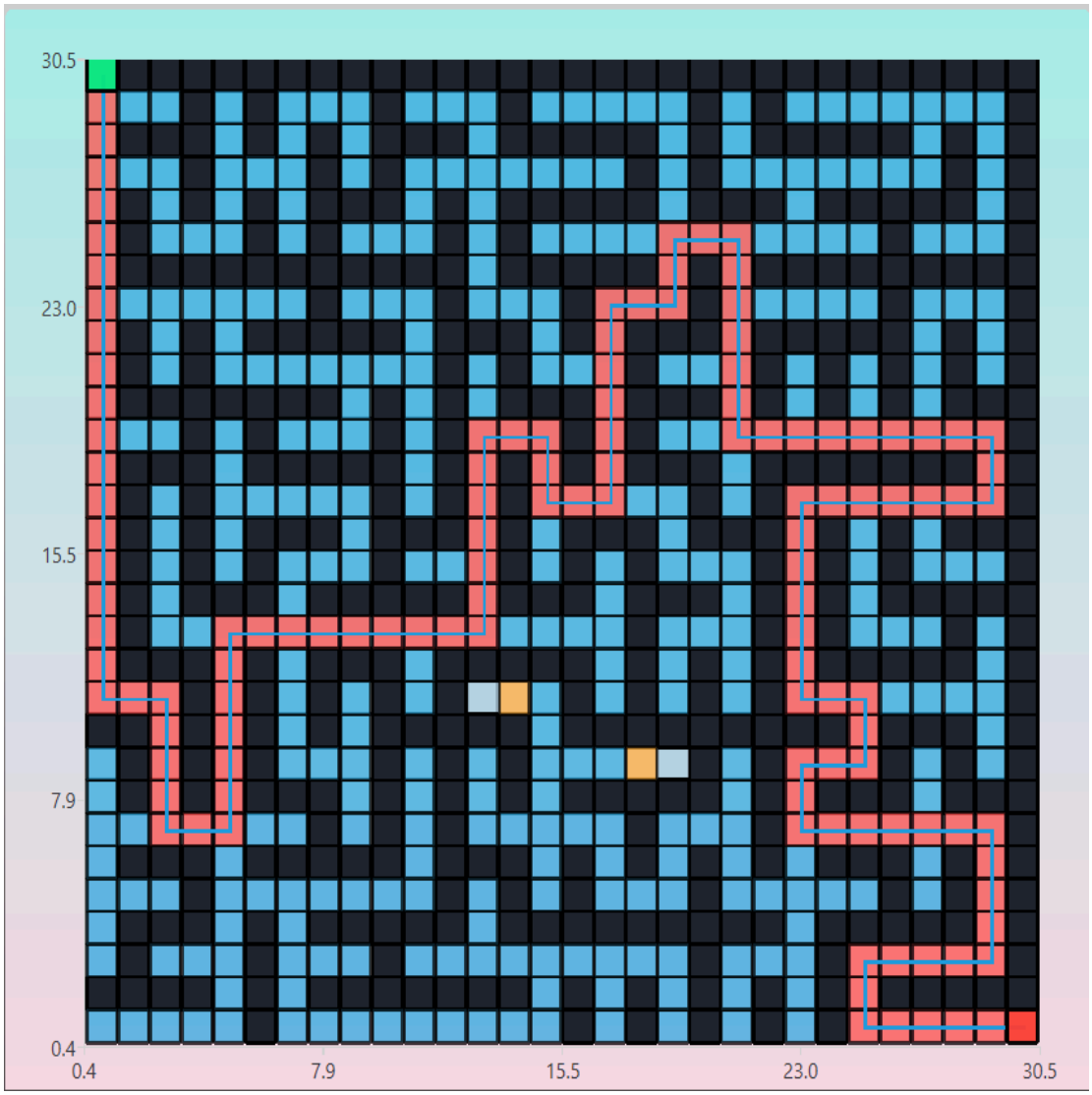

Figure A.2: Solved Maze using Pathfinding Algorithm

B File Structure

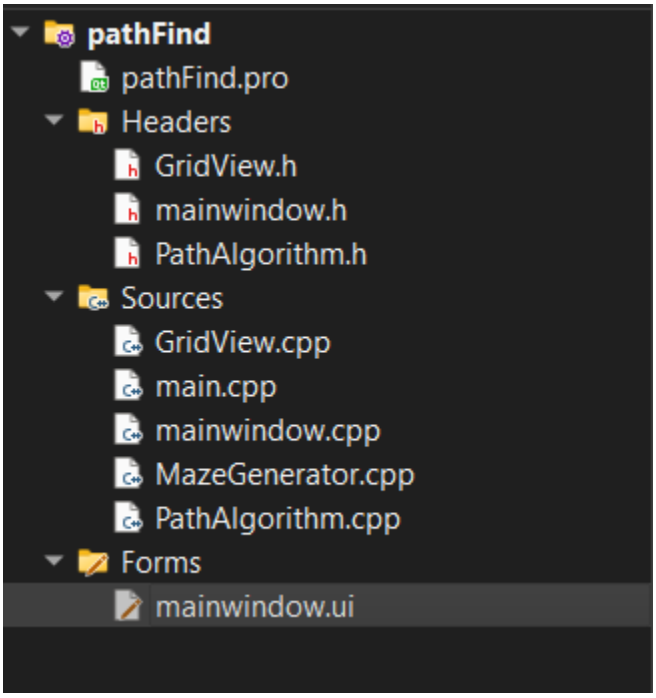

Figure B.1: File Structure of the Codebase