

**Kathmandu University**  
**Department of Computer Science and Engineering**  
**Dhulikhel, Kavre**



**A Project Report**  
**on**  
**“DSA Mini Project”**

**[Code No:COMP 202]**

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

**Submitted by**

**Mahesh Panta(33)(037987-24)**

**Asmit Panthi(34) (037988-24)**

**Sangam Paudel(36)(037989-24)**

**Submitted to**

**Sagar Acharya**

**Department of Computer Science and Engineering**

**Submission Date: February 22, 2026**

## **Bona fide Certificate**

**This project work on**

**“DSA Mini Project”**

**is the bona fide work of**

**“**

**Mahesh Panta(33)**

**Asmit Panthi (34)**

**Sangam Paudel (36)**

**”**

**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 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 .

We also extend our gratitude to our faculty members and staff for the positive atmosphere they created that accelerated our collaboration and productivity. Furthermore, we acknowledge and admire the support, motivation we received from our family members and our friends.

This mini project has helped us gain hands-on experience in problem-solving, teamwork, effective communication, and technical implementation. It has contributed significantly to our growth, both individually and collectively as a team.

**Mahesh Panta(33)**

**Asmit Panthi (34)**

**Sangam Paudel (36)**

## **Abstract**

Pathfinding is an important concept in computer engineering, often used in games, robotics, and navigation systems. This project has built a graphical tool that allows users to visualize in a maze generated. The tool was developed using C++ and Qt. Its primary focus was on static modes of mazes. In static mode, users can generate mazes automatically 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 Dijkstra's algorithm for pathfinding and Depth-First Search algorithm for maze generation. All of these are common graph algorithms. This helps users learn how pathfinding works and visualize .

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

## Table of Contents

<b>Acknowledgement</b>	<b>i</b>
<b>Abstract</b>	<b>ii</b>
<b>List of Figures</b>	<b>iv</b>
<b>Acronyms/Abbreviations</b>	<b>v</b>
<b>Chapter 1 Introduction</b>	<b>1</b>
1.1 Background	1
1.2 Objectives	1
1.3 Motivation and Significance	2
<b>Chapter 2 Literature Review</b>	<b>3</b>
<b>Chapter 3 Design and Implementation</b>	<b>4</b>
3.1 System Requirement Specifications	4
3.1.1 Software Specifications	4
3.1.2 Hardware Specifications	4
3.2 Implemented Algorithms	5
3.2.1 Algorithms used for Maze Generation	5
3.2.2 Algorithms used for Pathfinding	5
3.3 GUI Overview	6
3.4 System Diagrams	8
<b>Chapter 4 Discussion on the achievements</b>	<b>10</b>
4.1 Features	10
4.2 Challenges	10
<b>Chapter 5 Conclusion and Recommendation</b>	<b>11</b>
5.1 Limitations	11
5.2 Future Enhancement	11
<b>References</b>	<b>13</b>
<b>APPENDIX</b>	<b>14</b>

## List of Figures

Figure 3.1 Select Interaction Dropdown	6
Figure 3.2 Visualization Tab	7
Figure 3.3 Use-case diagram	8
Figure 3.4 System Workflow diagram	9

## **Acronyms/Abbreviations**

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

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:

- To allow users to generate maze in a grid using algorithms such as DFS.
- To create a user interface so that users can visualise BFS for maze creation and Dijkstra's algorithm for pathfinding, set parameters and control the simulation (pause/play).

### **1.3 Motivation and Significance**

The idea for the project came from the need to create an educational and visual tool for understanding pathfinding algorithms and maze generation algorithms. These algorithms are 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 also helped show how fast and efficient different algorithms are, and how they work with data in real time. Besides helping people learn, it can also be used to test how pathfinding works in simple, controlled setups. This can be useful for building more advanced systems in the future.

## **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
- **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

- **DFS (Recursive backtracking):** This algorithm starts with a grid filled with walls. A random cell is chosen as the starting point. From there, it moves to an unvisited neighbor two cells away, carving a path between them by removing the wall. Each move is pushed to a stack. If no unvisited neighbors remain, it backtracks using the stack until it finds one, continuing the process until all reachable cells are visited. The result is a maze with long, winding paths and minimal loops.

### 3.2.2 Algorithms used for Pathfinding

- **Dijkstra's Algorithm:** Dijkstra's algorithm finds the shortest path by expanding nodes in order of increasing path cost from the start. Each node keeps the minimal local cost found so far. Nodes are processed from a priority queue; neighbors are relaxed with updated costs if a shorter path is found. The algorithm terminates upon reaching the goal or exhausting nodes. Path reconstruction is done via parent pointers.

### 3.3 GUI Overview

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

#### 1. Simulation Tab

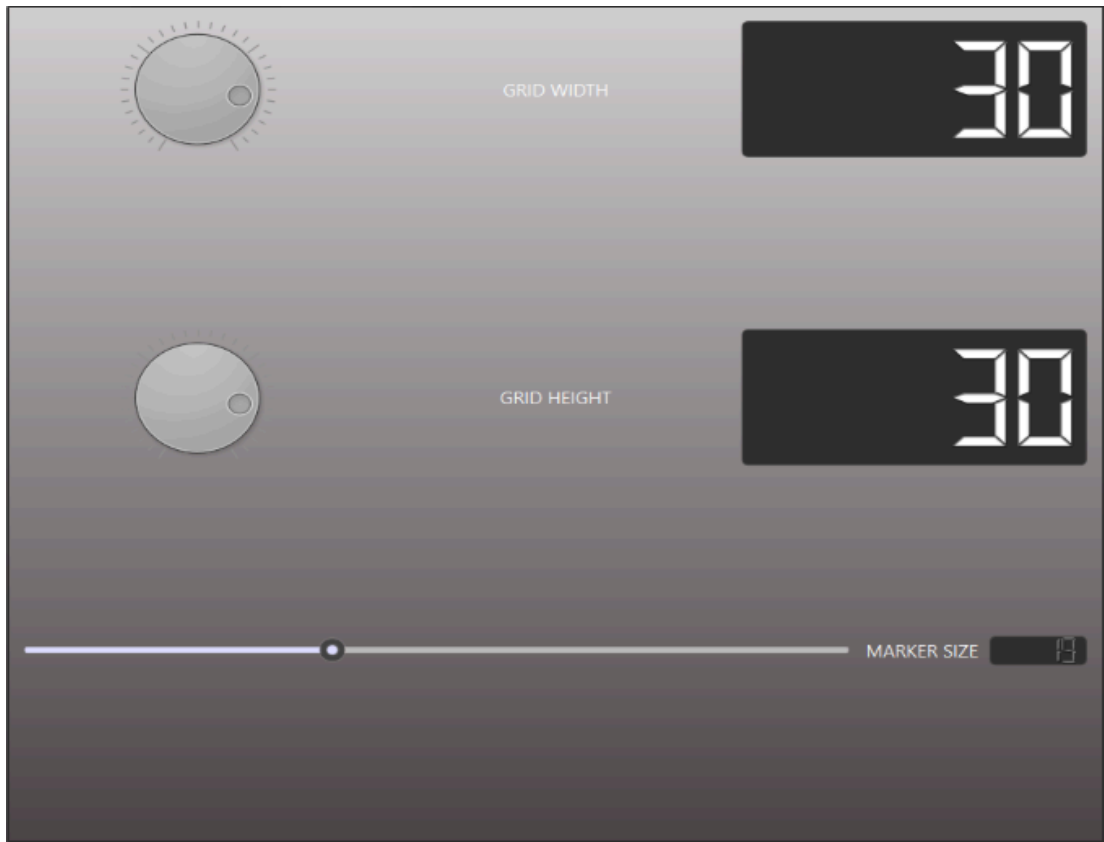
- A. Select pathfinding algorithm
- B. Set start and goal nodes



**Figure 3.1: Select Interaction Dropdown**

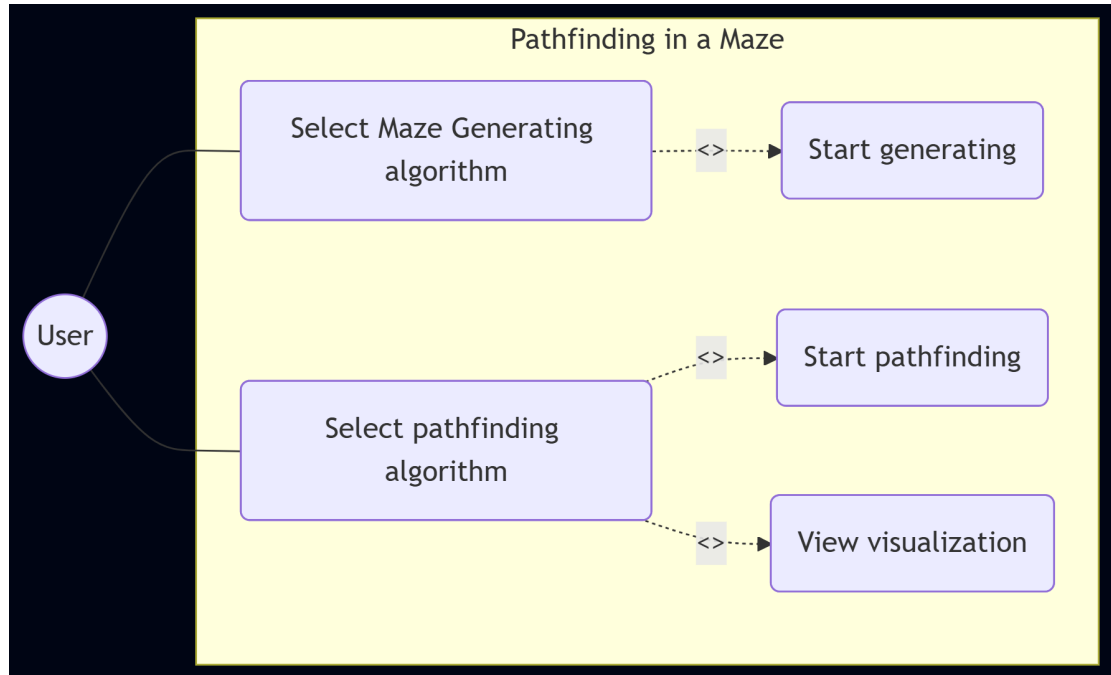
## 2. Visualization Tab

- A. Adjust horizontal and vertical node count
- B. Set Marker Size



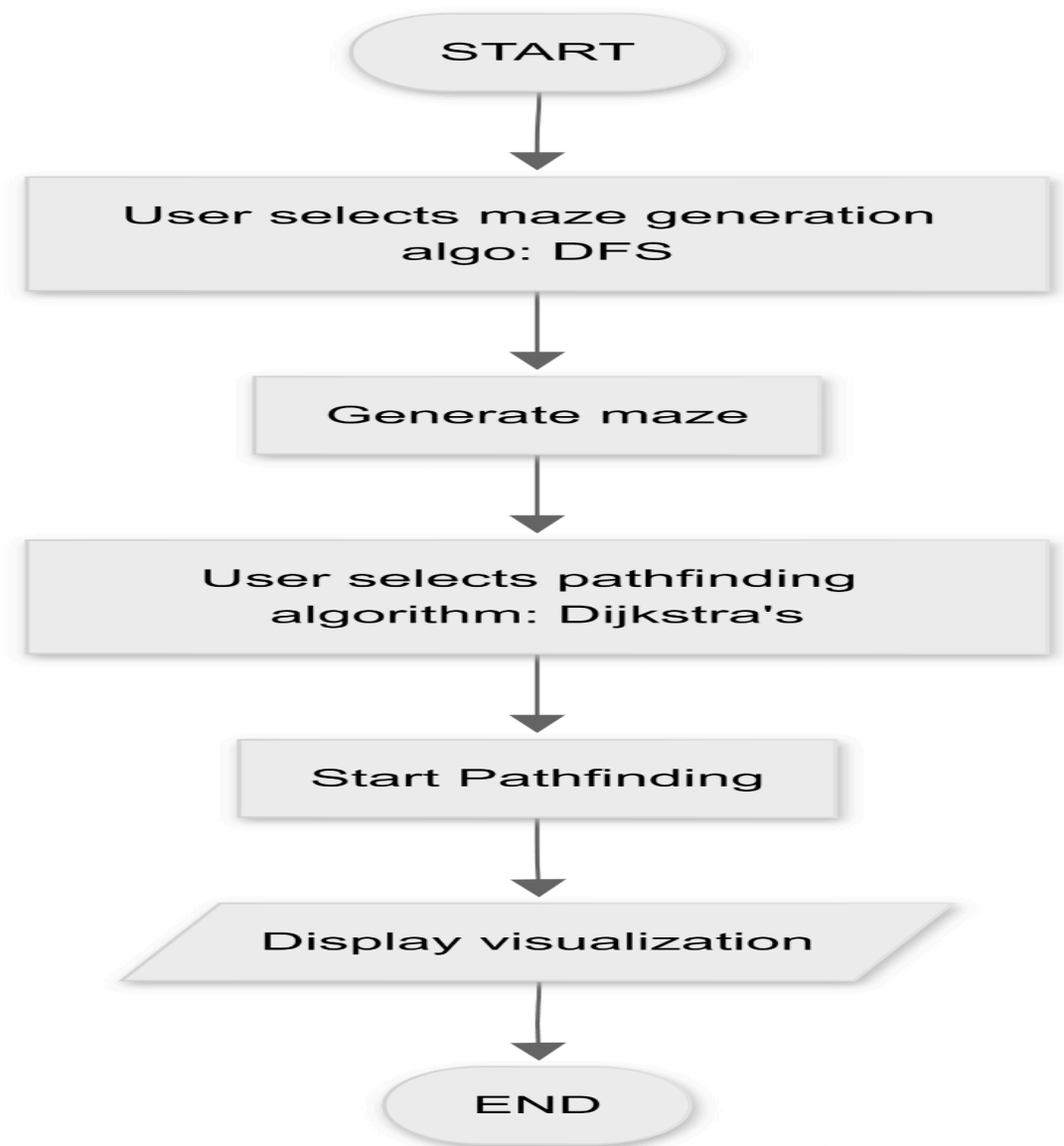
**Figure 3.2: Visualization Tab**

### 3.4 System Diagrams



**Figure 3.3: 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 three main use cases: selecting a maze generation algorithm, selecting a pathfinding algorithm, and analyzing performance logs. The "Select Maze Generating algorithm" use case includes the options to manually draw obstacles and start generating the maze. The "Select pathfinding algorithm" use case includes starting the pathfinding process and viewing the visualization.



**Figure 3.4: 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

The team has implemented several critical features enhancing both functionality and user experience. These include interactive simulation controls for visualization, both DFS and BFS algorithms on identical mazes. These additions collectively improve usability and extensibility of the system.

### 4.1      Features

Our team has successfully implemented a number of 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. Allows dynamic speed adjustment during runtime.

### 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 freezed during the visualization of the algorithms. The project required significant testing and debugging. Integrating manual controls with the existing automated pathfinding framework required resolving conflicts between user input and system-generated states, especially in maintaining consistent visual feedback and preventing unintended interactions.

## **Chapter 5      Conclusion and Recommendation**

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 grid. The real highlight, however, was to create a graphical interface that demonstrates the entire process, making the convoluted 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.

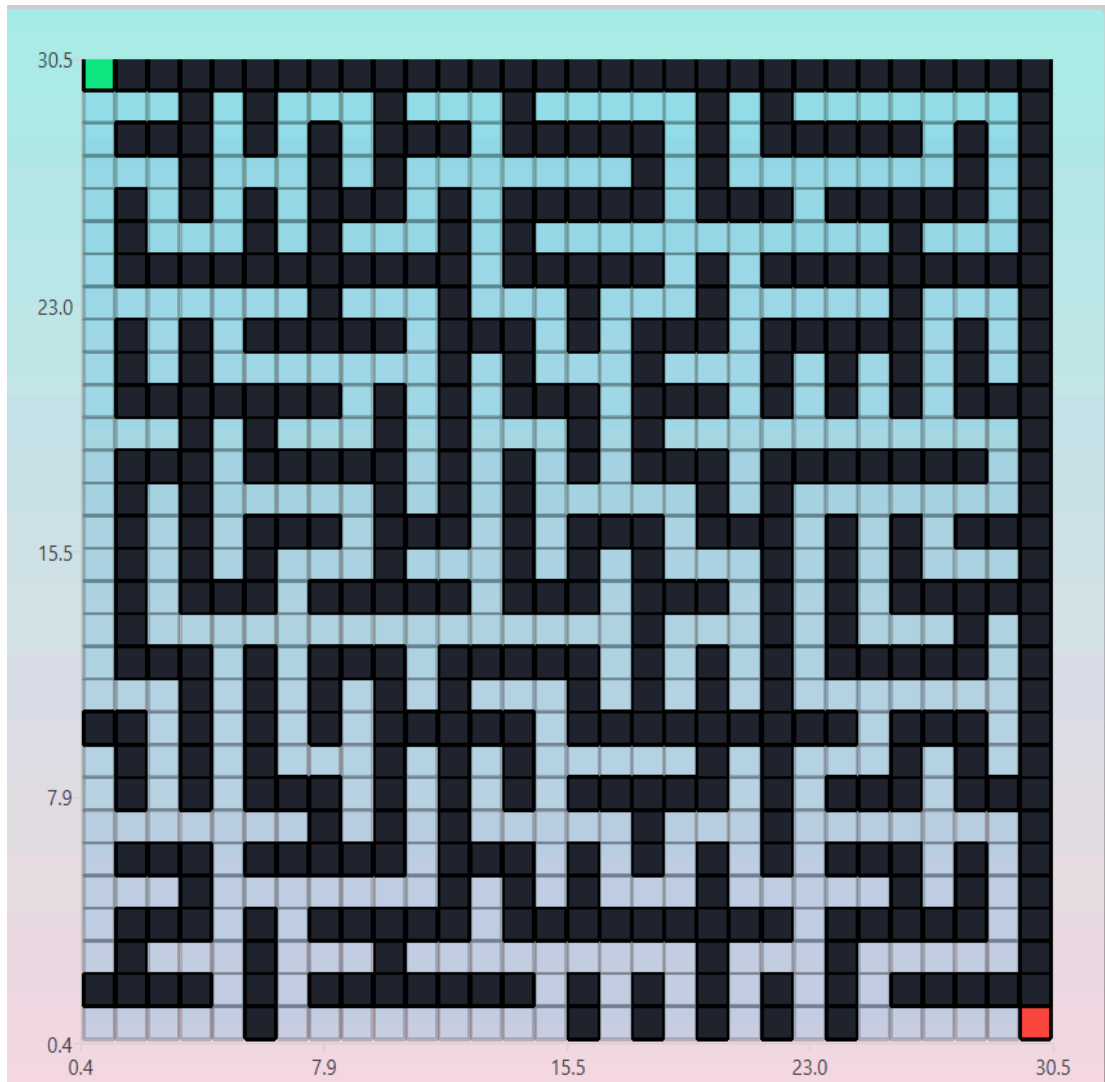
Permana, S. D. H., Bintoro, K. B. Y., Arifitama, B., & Syahputra, A. (2018). *Comparative analysis of pathfinding algorithms A\*, Dijkstra, and BFS on Maze Runner game*. *International Journal of Information System and Technology*, 1(2), 1–6.

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).

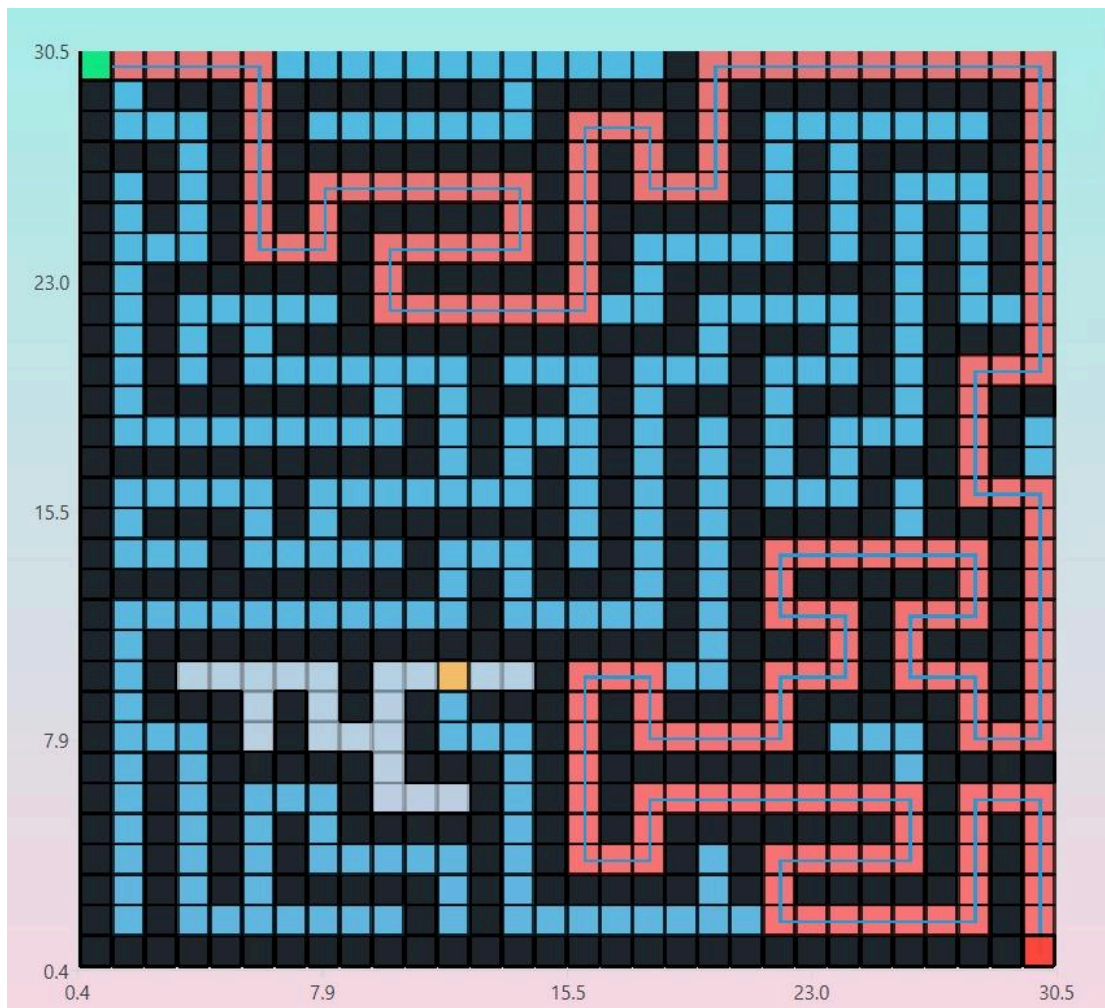
Muthuselvi, R., Sindhuja, A., & Sridevi, P. K. D. (2016). *Parallelization of maze generation and solving in multicore using OpenMP*. *International Journal of Innovative Research in Science, Engineering and Technology*, 5(Special Issue 11), 7–12.

## APPENDIX

### A Sample Outputs

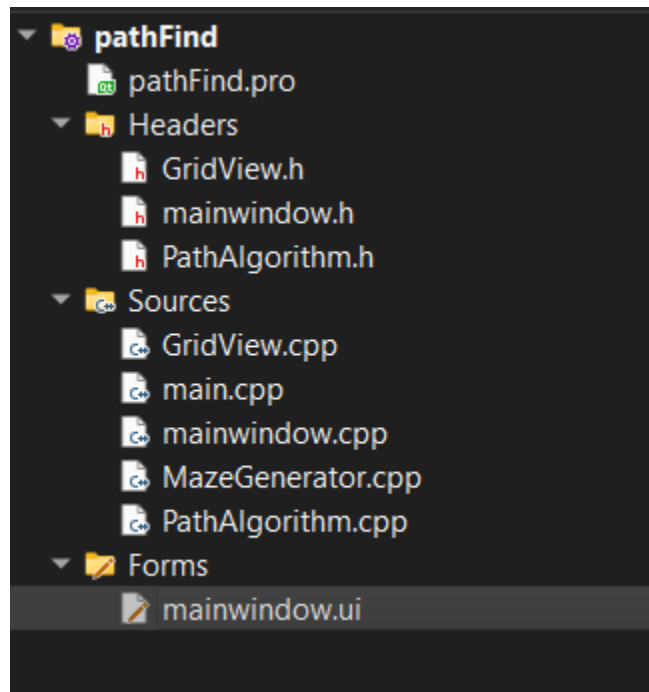


**Figure A.1: Generated Maze**



**Figure A.2: Solved Maze using Pathfinding Algorithm**

## B File Structure



**Figure B.1: File Structure of the Codebase**