MAZE SOLVER USING DFS

ENPM809Y - Introductory Robot Programming

Group 4

Karan Sutradhar, Sudharsan Balasubramani, Sai Praveen Bhamidipati, Ashwin Prabhakaran

INTRODUCTION

- **Objective :**

  We implemented a program that computes a path given the current position/ starting position (S) and orientation (fixed in this case) of a robot in a maze to reach the center of the maze (G).

- **Method :**

  Path Planning Algorithm is Depth First Search Algorithm (DFS)

CONSTRAINTS/ ASSUMPTIONS

- The Robot doesn’t know the location of the wall unless it reaches near it.
- MMS Simulator interface.
- Selection of maze from given mazes (Discretized mazes).
- Robot transverse in only x-y direction (2D plane).
- Robot can only move in forward direction and have holonomic behavior, it cannot move in arbitrary direction.
- Action space of robot is right, left and Forward.

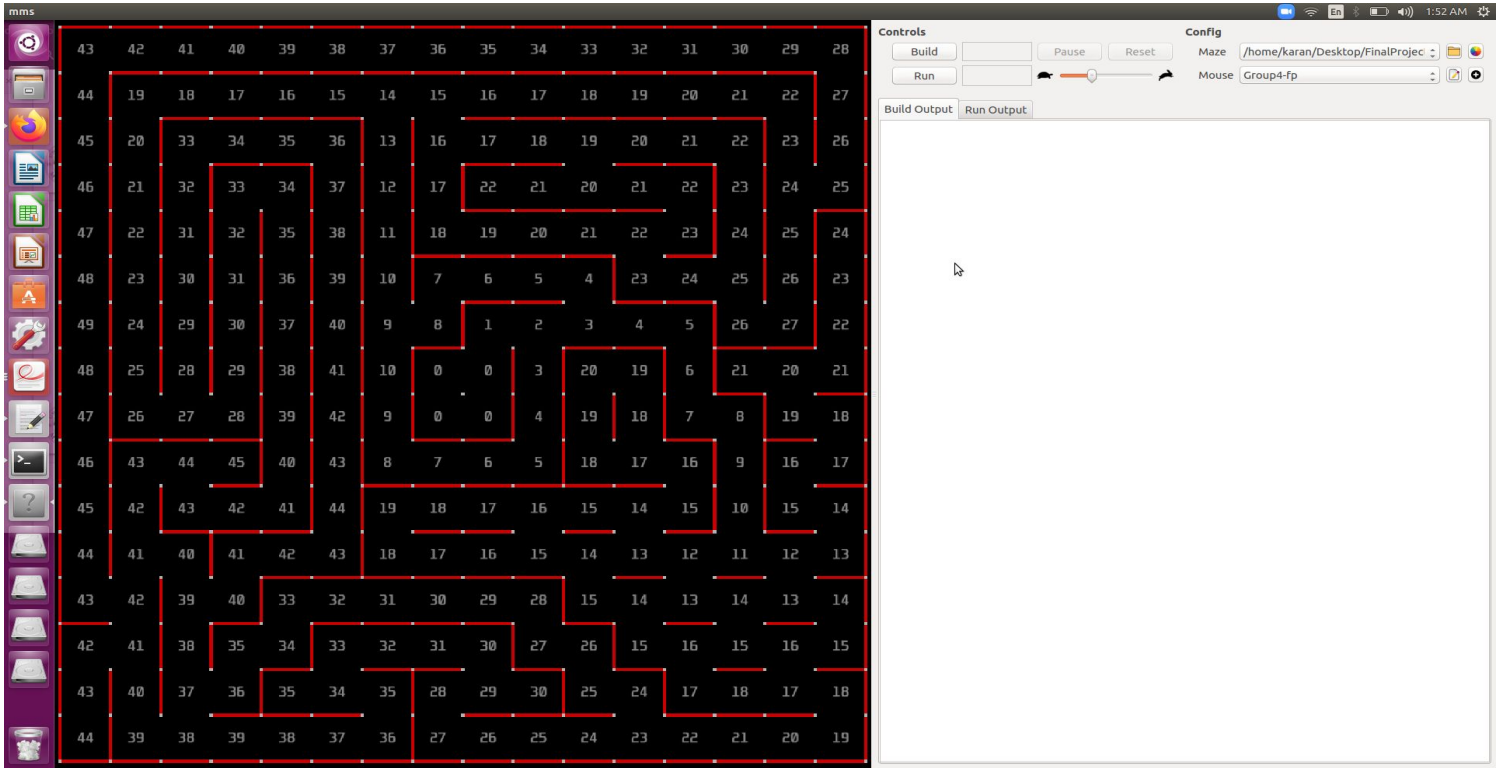
LEARNING OUTCOMES

- Object-oriented programming, inheritance, and dynamic polymorphism.
- Implementation of Path Planning Algorithm (DFS) using dynamic programming.
- Usage of Data Structures.

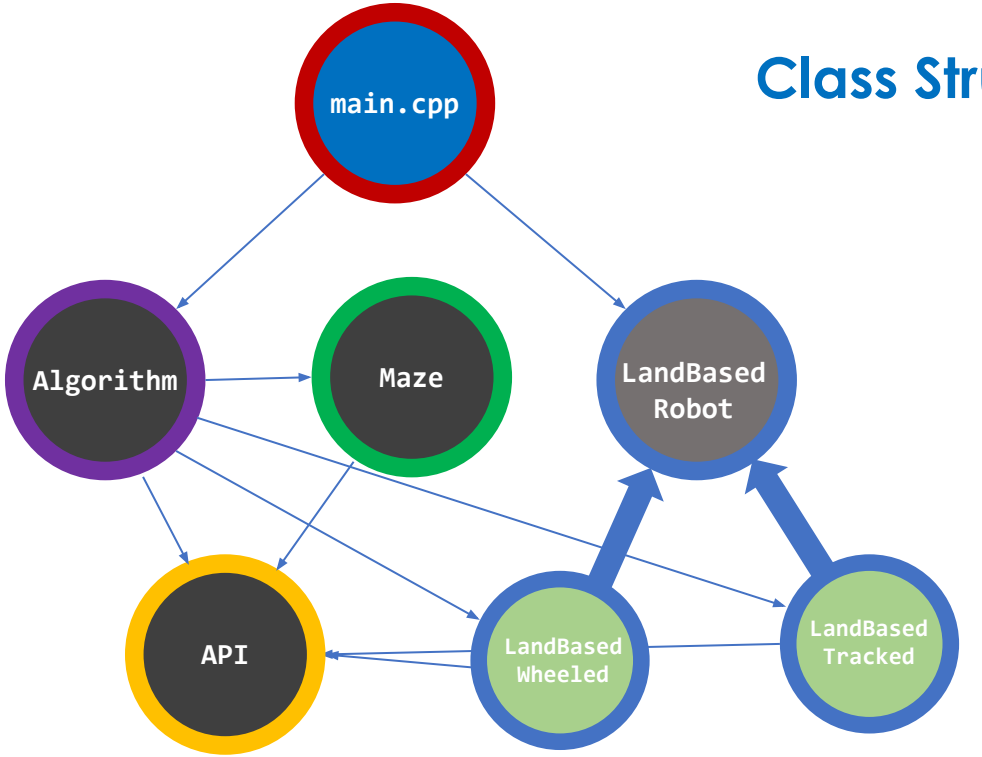
METHODOLOGY

- OOP based programming.
- Simulator interface - API class.
- Reading maze from given set of mazes and dynamic implementation of path planning algorithm (DFS - which doesn’t give the cost effective or optimal path).

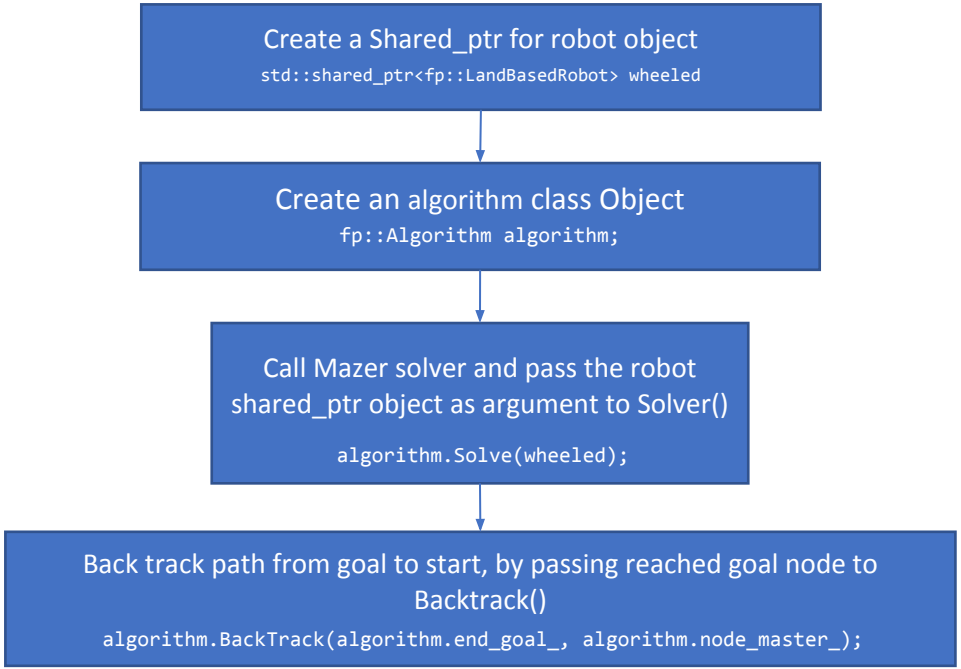
MMS SIMULATOR



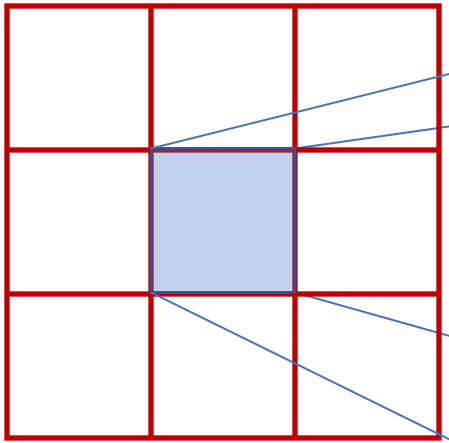
Class Structure



PROCESS FLOW



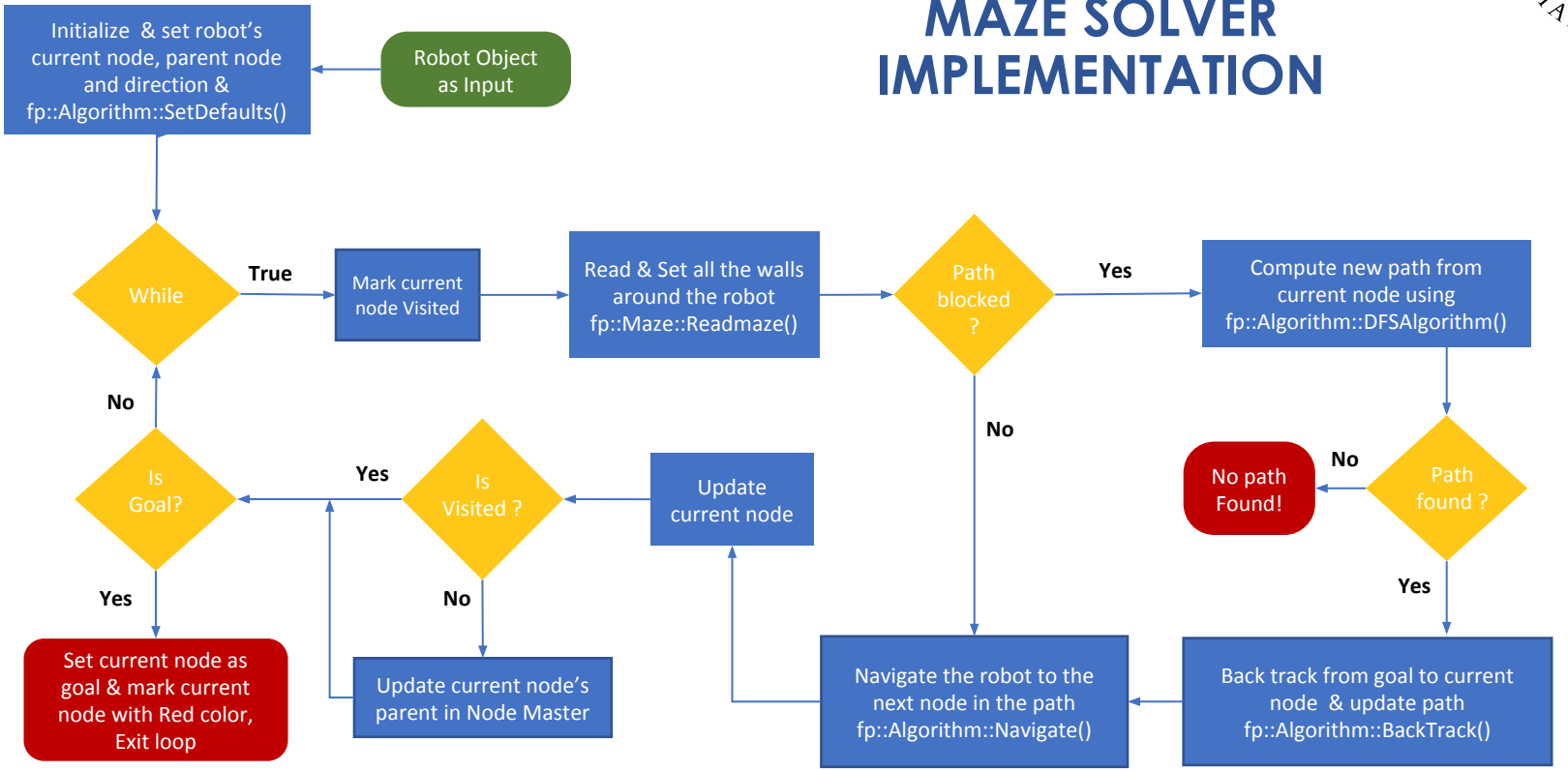
NODE DATA STRUCTURE



```
struct Node {

    //---> Attributes <---//
    std::array<int, 2> parent_node_;

    //---> Default constructor <---//
    Node(): parent_node_{}{}

    //---> Destructors <---//
    ~Node()= default;.

};
```

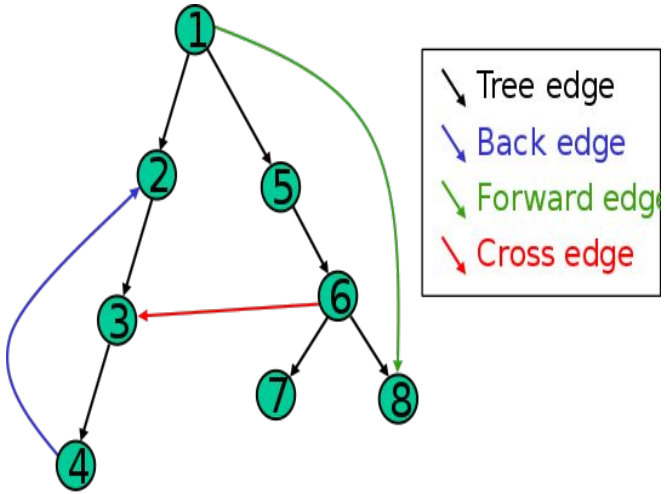
std::array<std::array<Node, 3>,3> node_info_;

MAZE SOLVER IMPLEMENTATION



DEPTH-FIRST SEARCH

- Depth-First Search (DFS) is an algorithm for traversing or searching tree or graph data structures.
- The algorithm starts by selecting some arbitrary node as the root node and explores as far as possible along each branch.



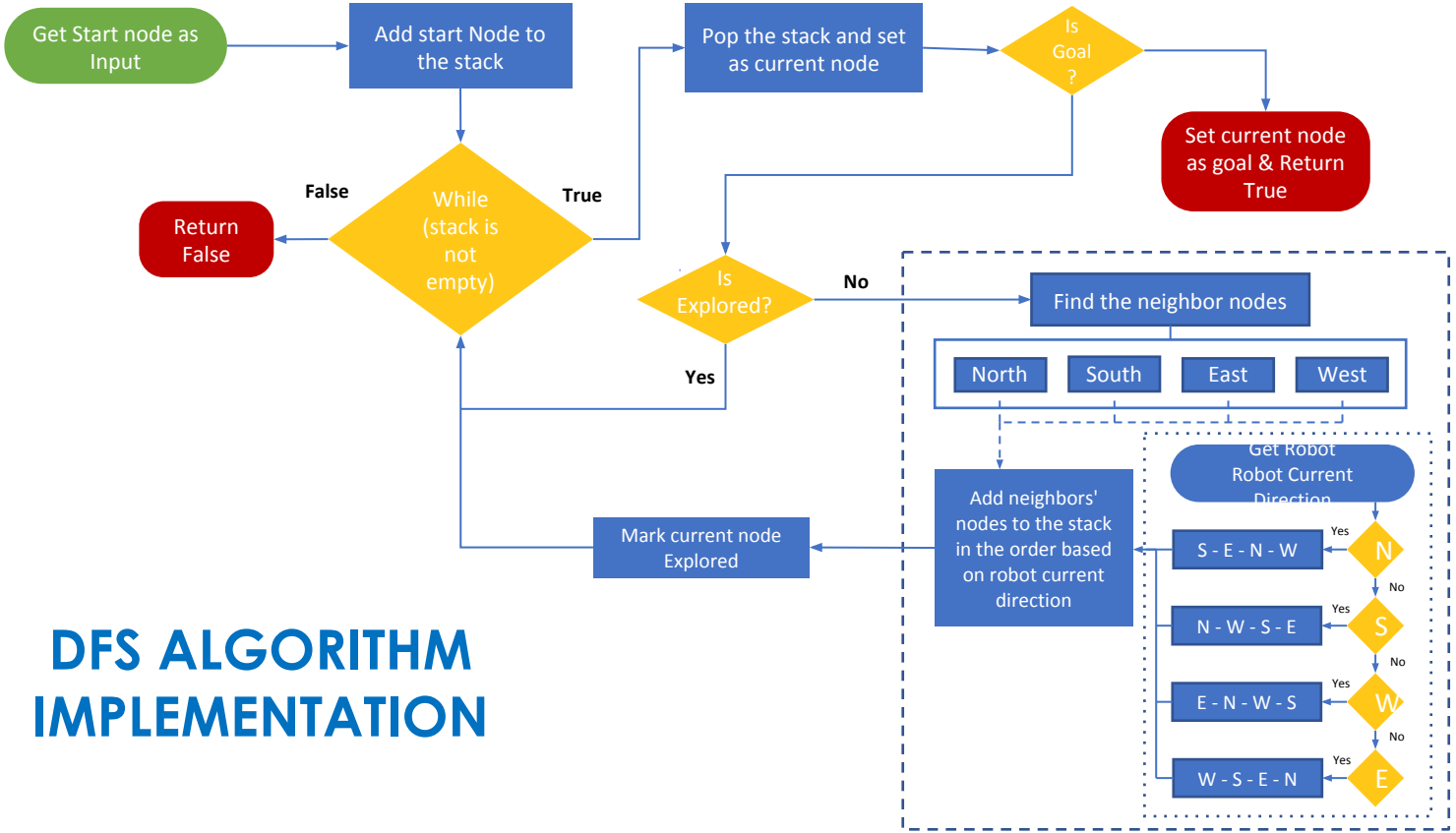
Credits: https://en.wikipedia.org/wiki/Depth-first_search

DEPTH-FIRST SEARCH

- The implementation of DFS is similar to that of BFS, but differs in two ways,
  - It uses stack instead of queue.
  - It delays checking whether a vertex has been discovered until the vertex is popped from the stack rather than making this check before adding the vertex.

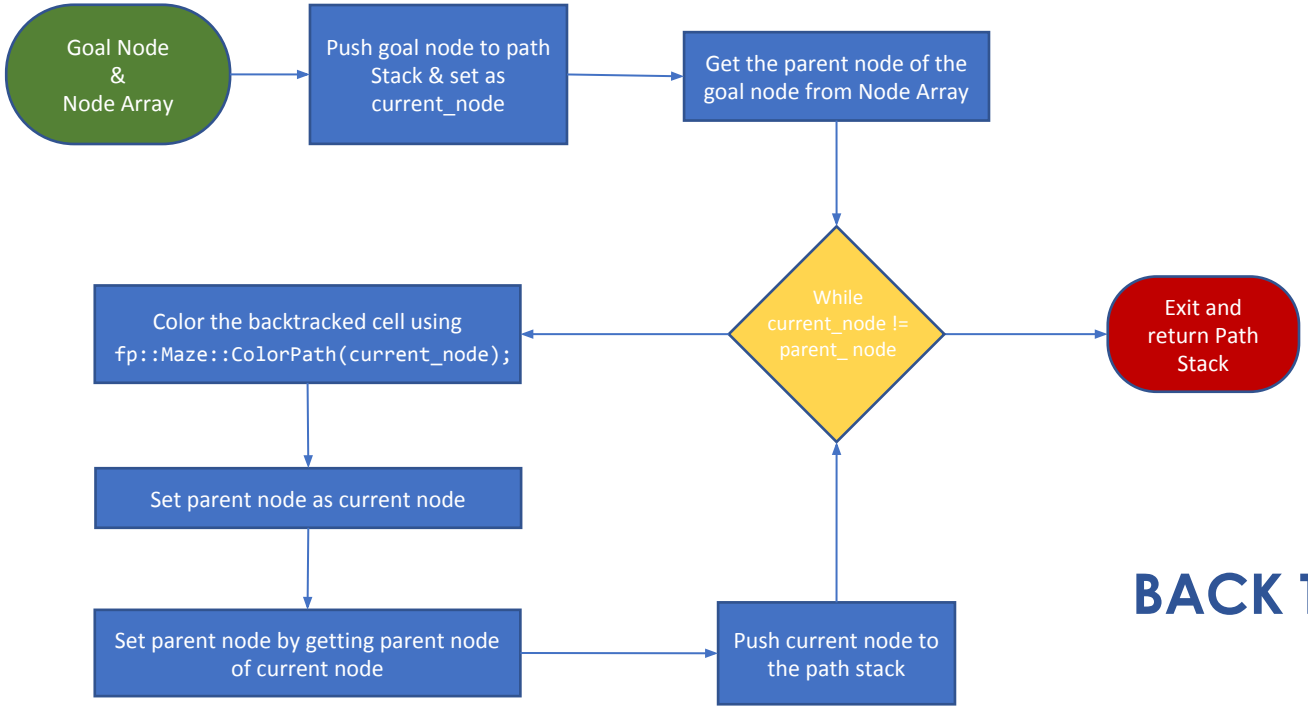
- After the goal is reached, it shows a path from the start node to the goal node.

DFS ALGORITHM IMPLEMENTATION



BACKTRACKING

- Backtracking means that when there are no new nodes to move forward by the algorithm, it traverses back on the same node to find the path.
- In order to back track, a parent node data type is introduced which matches to its respective child nodes.
- Also a visited data set is used, so that to keep track of the nodes which has already been seen by the robot. This prevents from duplicating a node. Only the new nodes would be added onto the stack.
- The backtracking will be done based on the cost map to obtain optimal solution, whichever path is the least from the start to goal node based on the cost function, the backtracking takes place.



BACK TRACKING

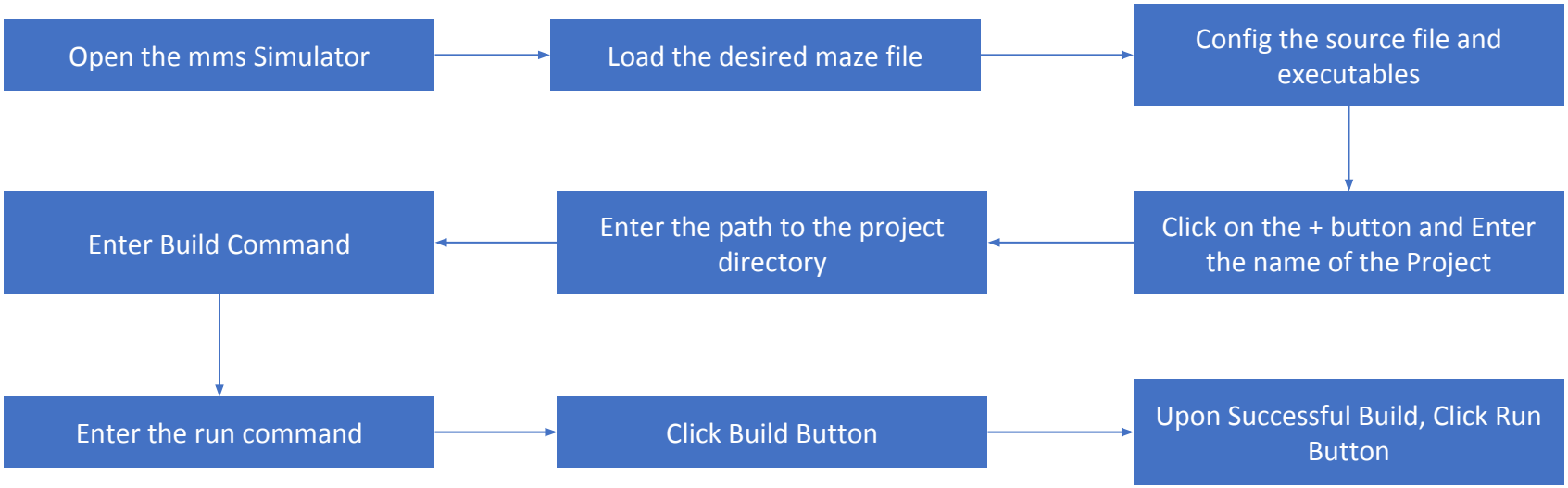
MMS SIMULATOR INSTALLATION

Visit this Github link https://github.com/mackorone/mms#building-from-source to install Micromouse simulator.

Steps to run this program:-

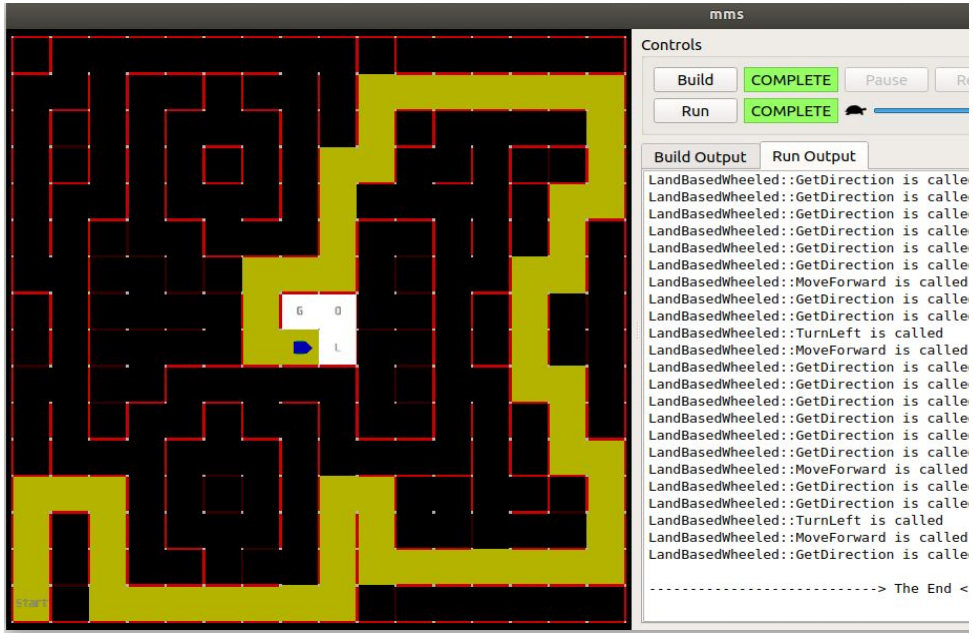
1. Build the simulator using the following command: /user# cd mms/src /user# qmake && make
2. Run the simulator using the following command: ../bin/mms
3. Parameters to be changed in the simulator:- Name:- Input your desired name. Directory:- ../../Final-Project-Group4/Final_Project/src Build Command:- g++ -std=c++14 API/api.cpp LandBasedRobot/landbasedrobot.cpp LandBasedTracked/landbasedtracked.cpp LandBasedWheeled/landbasedwheeled.cpp Maze/maze.cpp Algorithm/algorithm.cpp main.cpp Run Command:- ./a.out
4. Run the Simulator.

PROJECT EXECUTION



RESULTS

- Successfully implemented DFS Algorithm to solve the maze.
- OOP based code development.



FUTURE IMPROVEMENTS

- We would like to implement other path planning algorithms like BFS, Dijkstra, A*, etc which runs faster and give optimal results for different and difficult mazes.
- Changing the Micromouse Simulator to add dynamic obstacles on already explored locations.

REFERENCES

- **DFS Algorithm** : https://www.tutorialspoint.com/data_structures_algorithms/depth_first_traversal.htm
- **BFS Algorithm** : https://en.wikipedia.org/wiki/Breadth-first_search
- **Mms simulator** : https://github.com/mackorone/mms
- **Object Oriented Programming** : https://beginnersbook.com/2017/08/cpp-oops-concepts/

CONTRIBUTIONS

Karan Sutradhar - Blueprint of Project/ Work Allotment and Coordinating-Setting up zoom meetings-Project timeline/Coding-DFS and Class Structure/ Presentation.

Sudharsan - POC with Professor for Doubts/Coding-DFS and Class Structure/ Compiling Codes/ Error Handling and Testing/ Flow Charts for Presentation.

Sai Praveen Bhamidipati - Background data collection/ Doxygen file/ Coding- Maze and API classes/ Documentation- comments on codes/ Report and Presentation/Error Handling/ Testing.

Ashwin Prabhakaran - Github Repository Maintenance/ Verifying Google Naming Conventions/ Coding- Maze and API classes/ Documentation- comments on codes/ Report/ Presentation/Testing.

THANK YOU

ANY QUESTIONS?