# Instructions

This program has both the command line interface and a graphical user interface that allow the user to create new maze and visualize the path finding process. In order to use the command line interface, the user use the command: search <filename> <method> where filename is the path to a .txt file that specifies the maze configuration (including start, goals, and wall points). Method is the search algorithm that will be used in the path finding process. There are 6 different methods, including Breadth-first search (BFS), Depth-first search (DFS), Greedy Best first search (GBFS), A Star Search (AS), Bidirectional Search (CUS1), Iterative Deepening A Star Search (CUS2) (Note: you only need to specify the abbreviation inside the brackets, not the full name). If the arguments are valid, the program will run the searching process and produce the following output:

filename method number\_of\_nodes  
path

where filename is the path to the .txt file, method is the search algorithm, number\_of\_nodes is the ????, and path is either a sequence of moves in the solution to reach one goal from the start location or the message “No solution found” if there is no solution.

On the other hand, the user can use the graphical interface to create a new maze.txt file and see the searching process visually. To create a new maze, the user have to speicified the row, column, start, goals, and walls points. To select a point in a maze, the user can simply click on a square in the maze and press the confirm button. Additionally, for the goals and walls, you can select multiple squares and remove selected squares by clicking on the already selected squares

A picture containing crossword puzzle, text, appliance, fruit

Description automatically generated

After creating the maze, the user can save the file and export it to a location in the computer. Next, the user can go back to the homepage to import the maze.txt file that are just created and start the searching process. Finally, in the search page, the user can select one of 6 algorithms to see the searching process.

A picture containing website

Description automatically generated

Them legend

# Introduction

In the Robot Navigation problem, the environment is an N x M grid where N and M are both greater than 1. Moreover, there are a number of occupied wall cell that can not be reached. A robot is initally located in one of the cell and it is required to avoid the walls and find a path to reach one of the designated goals. To find a solution for this problem, we can use several search algorithm such as Breadth-first search and Depth-first search in order to find a correct path. Therefore, in this report, we will look at some common algorithms that has been used in the program and compare them based on their time complexity, memory.

# Search Algorithms

## Uninformed search algorithms

Uninformed search algorithms are basic algorithms that can find a solution in a search tree using a brute-force method without using any domain-specific knowledge to guide the search. Some common uninformed algorithms are Breadth-first search, Depth-first search, and Bidirectional Search.

Breadth-first search (BFS): In BFS, the algorithm will start at the root node and keep expanding the shallowest unexpanded node until it finds a solution. Therefore, it will check all nodes at the same depth level before moving on to explore the nodes at the next depth. As a result, BFS is can find the shortest path from the root node to the solution if the actions are unweighted. Furthermore, BFS store the nodes on the queue in a first-in first-out principle to ensure the shallowest node is explored first. Because the breadth-first search will have to explore every nodes in the worst case; therefore its time complexity is *O*(*bd*+ 1)

,where b is the branching factor, and d is the distance from the start node. Moreover, because the BFS need to store all the nodes at the same depth; therefore, the memory consumption will be huge if the maximum width in the graph is large.

Depth-first search (DFS): Instead of exploring the shallowest unexpanded node, the DFS will check the deepest unexpanded node. The algorithm will start at the root node and recursively explore all the children node along the same branch as far as possible before backtracking and check different branches. As a result, the found solution is not guaranteed to be the shortest path from the root node to the solution. To implement DFS, we can use a stack data structure in a last-in first-out faction to make sure the deepest node is explored first.

## Informed search algorithms

References:

<https://en.wikipedia.org/wiki/Breadth-first_search>

<https://www.geeksforgeeks.org/illustrate-the-difference-in-peak-memory-consumption-between-dfs-and-bfs/>

https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/