COS30019
INTRODUCTION TO
ARTIFICIAL INTELLIGENCE
(HANOI)

Assignment 1:
Tree Based Search
Option B - Robot Navigation

GIA HUY NGUYEN
103441107
SWH00039

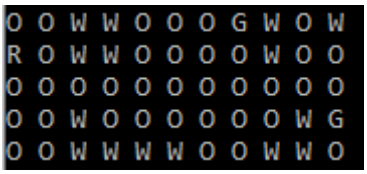
Semester May 2023

Table of contents

I. Introduction

1. Problem

Robot Navigation Problem: The environment is an NxM grid (where N > 1 and M > 1) with several walls occupying some cells (marked as “W”). The robot is initially located in one of the empty cells (marked as “R”) and required tofind a path to visit one of the designated cells of the grid (marked as “G”).



2. Related Theory Base

- The concept of a Tree holds great significance in Graph theory, as well as in the data structures and algorithms. A tree is a graph in which any two vertices are connected by exactly one path. In other terms, any connected graph without cycles is a tree. Additionally, a forest is a disjoint union of trees. Trees find extensive application in computer science data structures, including binary trees, heaps, trie, Huffman trees for data compression, and more.
- To classify a graph as a tree, there are essential conditions that need to be met.

Let G=(V,E) graph with n vertices.

The following six propositions areequivalent:

+ G is a tree.
+ G has no cycles and has n-1 edges.
+ G is connected and has n-1 edges.
+ G has no cycle and if an edge is added to join two non-adjacentvertices, a unique cycle occurs.
+ G is connected and if any edge is removed, G loses its connectivity.
+ Each pair of vertices in G is connected by a unique path.

- A tree (with roots) is a tree in which a vertex is chosen as the root and eachedge is oriented to coincide with the direction of a single path from the root to each vertex.
- Binary tree:

+ A tree with roots each vertex has no more than two children is called abinary tree (*binary tree*).
+ A binary tree in which each vertex has exactly two children is called afull binary tree (*full binary tree*)
+ A binary tree where all leaves have the same level is called a perfectbinary tree (*perfect binary tree*).
+ A binary tree whose every vertex has a right child also has a left child iscalled a near-complete binary tree (*almost complete binary tree*).

+ A binary tree with a leaf level difference of no more than 1 level (the height of the tree) is called a balanced binary tree.

3. Search Methods

Search Type	Description	Method
Uninformed		
depth-first search	Select one option, try it, go back when there are no more options	DFS
breadth-first search	Expand all options one level at a time	BFS
Informed		
greedy best-first	Use only the cost to reach the goal from the current node to evaluate the node	GBFS
A* ("A Star")	Use both the cost to reach the goal from the current node and the cost to reach this node to evaluate the node	AS
Custom		
Your search strategy 1	An **uninformed** method to find a path to reach the goal.	CUS1
Your search strategy 2	An **informed** method to find a shortest path (with least moves) to reach the goal.	CUS2

II. Search methods

1. Depth-first search(DFS)

a. Theory

- Depth-first search (DFS) is an algorithm used to explore or traverse a tree or a graph. The algorithm commences at the root (or a selected vertex) and grows as far as possible on each branch.
- Typically, DFS operates as an incomplete information search, where it continues searching by moving to the first child vertex of the current node until it reaches the desired vertex or encounters a node without any children. Subsequently, the algorithm backtracks to the previously visited vertex. In the non-recursive form, all pending vertices to be explored are stored in a LIFO stack.
- DFS has a lower space complexity compared to BFS (breadth-first search). The time complexity of both algorithms is equivalent and denoted as O(|V| + |E|).

b. Algorithm

- Performing depth-first search (DFS) on an undirected graph can be likened to navigating a maze armed with a spool of thread and a bucket of red paint to mark your path and prevent getting lost. In this analogy, each vertex 's' in the graph corresponds to a door in the maze.
- To initiate the search, we select vertex 's,' tie the end of the thread to it, and label it as "visited." We then designate 's' as the current vertex, denoted as 'u'.
- Now, we proceed by following any edge (u, v).
- If the edge (u, v) leads us to a vertex 'v' that has already been marked as "visited," we return to vertex 'u'.
- If v is a new vertex, we move to v and roll the thread along. We label 'v' as "visited," make it the new current vertex, and repeat these steps.

- Finally, we may reach a vertex where all neighboring edges lead to "visited" vertices. In such cases, we retrace our steps by rewinding the thread until we return to a vertex adjacent to an unexplored edge. We resume the process of discovery as described above.
- The DFS procedure concludes when we return to vertex 's,' and there are no unexplored edges adjacent to it.

c. Pseudo Code

```
DFS(G, u)
    u.visited = true
    for each v ∈ G.Adj[u]
        if v.visited == false
            DFS(G,v)
init() {
    For each u ∈ G
        u.visited = false
     For each u ∈ G
       DFS(G, u)
}
```

2. Breadth-first search(BFS)

a. Theory

In graph theory, the breadth-first search (BFS) algorithm is employed to explore a graph by carrying out two operations: (a) given a vertex of the graph, (b) add vertices adjacent to the given vertex to the next reachable list. The breadth-first search algorithm serves two purposes: finding the path from an initial root vertex to a designated destination vertex, as well as determining the paths from the starting vertex to all other vertices within the graph. Notably, in unweighted graphs, breadth-first search always finds the shortest possible path. The BFS algorithm initiates from the root vertex and systematically investigates the neighboring vertices connected to the root vertex. Subsequently, for each of these vertices, the algorithm proceeds to explore their respective adjacent vertices that have not been previously observed, and the process is repeated accordingly. It is worth mentioning the depth-first search algorithm, which employs the same two operations but follows a distinct sequence in observing vertices compared to the breadth-first search algorithm.
This algorithm finds applications in the field of artificial intelligence. It utilizes a queue as the underlying data structure for implementation.

b. Algorithm

The algorithm employs a queue data structure to store intermediate information acquired during the search process:
- The original vertex is inserted into the queue (incoming).
- Get the first vertex in the queue and observe it

+ If this vertex corresponds to the target vertex, the search is halted, and the result is returned.
+ If not, all adjacent vertices to the visited vertex, which have not been observed before, are inserted into the queue.
- If the queue becomes empty, it signifies that all reachable vertices have been observed. At this point, the search concludes, and the result "not found" is returned.
- If the queue is not empty, the process returns to step 2.

Space complexity: If V is the set of vertices of the graph and |V| is the numberof vertices, the required space of the algorithm is O(|V|).

Time Complex Graph: If V, and E are the set of vertices and arcs of the graph,then the execution time of the algorithm is O(|V| + |E|) because in the worst case each vertex and the arc of the graph is visited exactly once.

c. Pseudo Code

```
Input: s as the source node

BFS (G, s)
let Q be queue.
 Q.enqueue( s )

mark s as visited
while ( Q is not empty)
v = Q.dequeue( )

for all neighbors w of v in Graph G
if w is not visited
Q.enqueue( w )
mark w as visited
```

3. Greedy best-first (GBFS)

a. Theory

By employing the greedy best-first algorithm, the first inheritance of the parent is expanded. Once the successor is generated, the following steps are taken:

- If the successor's heuristic is better than its parent, the successor is placed atthe front of the queue (with the origin being re-inserted right after it) and the loop restarts.
- Otherwise, the next piece is inserted into the queue (in a position determinedby its heuristic value). The process will evaluate the remaining successors (if any) of the parents.

b. Algorithm

The algorithm utilizes a queue data structure to keep track of intermediate information during the search process. GBFS prioritizes the exploration of vertices based on a heuristic evaluation rather than their order of discovery. The algorithm follows these steps:

- Initialize an empty priority queue.
- Insert the starting vertex into the priority queue.
- While the priority queue is not empty, do the following:
+ Dequeue the vertex with the highest priority (based on the heuristic evaluation) from the priority queue.
+ If this vertex corresponds to the target vertex, terminate the search and return the result.
+ Otherwise, mark the dequeued vertex as visited.
+ Generate a list of adjacent vertices to the visited vertex.
+ For each adjacent vertex not yet visited, calculate its heuristic value based on a predetermined heuristic function.
+ Insert the adjacent vertices into the priority queue according to their heuristic values, with the highest priority being given to the vertices with the lowest heuristic values.
- If the priority queue becomes empty before finding the target vertex, it indicates that no path exists. Terminate the search and return "not found."

The algorithm has a space complexity of O(|V|) since it only requires storing the visited vertices. However, the time complexity varies depending on the heuristic function used. Let's assume h(v) represents the heuristic value for vertex v.

The worst-case time complexity of GBFS can be stated as O(|V| + |E| * h_max), where h_max represents the maximum heuristic value among all vertices. This complexity arises when all vertices are enqueued and dequeued from the priority queue, with each dequeue operation taking O(1) time and the heuristic evaluation taking O(h_max) time.

c. Pseudo Code

```
procedure GBS(start, target) is:
  mark start as visited
  add start to queue
  while queue is not empty do:
    current_node ← vertex of queue with min distance to target
    remove current_node from queue
    foreach neighbor n of current_node do:
      if n not in visited then:
        if n is target:
          return n
        else:
          mark n as visited
          add n to queue
  return failure
```

4. A* (“A Star” - AS)

a. Theory

A* is an algorithm used for graph search, specifically to find a path from a starting node to a given destination node or to a node satisfying a destination condition. This algorithm uses a "heuristic evaluation" to rank each node according to its estimate of the best route through that node. This algorithm traverses the nodes in the order of this heuristic evaluation. Therefore, the A* algorithm is an example of best-first search.

b. Algorithm

When it comes to solving the pathfinding problem, which A* is commonly used for, the algorithm gradually constructs all routes from the starting point until it finds a path that reaches the destination. However, similar to all informational search algorithms (informed search algorithms), A* only builds routes that "seem" to lead towards the destination.

To determine which routes are more likely to lead to the destination, A* uses a "heuristic evaluation" of the distance from any given point to the destination. In the case of pathfinding, this estimate could be the crow's way distance - a commonly used approximation for the distance of a roadway.

One distinction of A* compared to best-first search is that it also accounts for the distance traveled. That makes A* "complete" and "optimal", that is, A* will always find the shortest path if such a path exists. A* is not guaranteed to outperform simpler search algorithms. In a maze-like environment, the only way to get to the destination might be to first go further away from the destination and then finally come back. Consequently, trying the nodes in the order of "closer to the destination, first try" can consume significant time and effort.

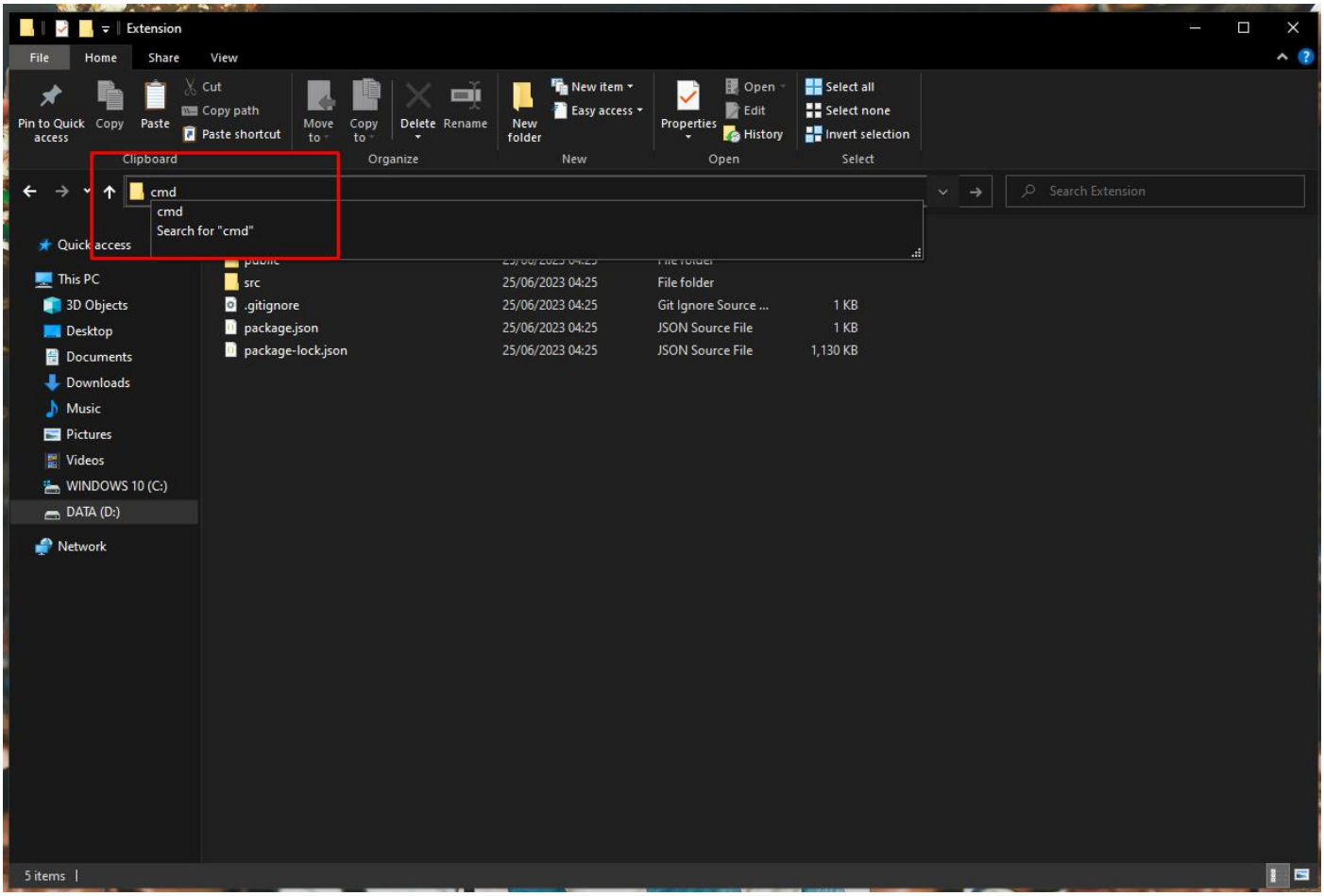
c. Pseudo Code

```
function A*(start,goal)
    closedset := the empty set
                                    // The set of nodes already evaluated.

    openset := {start}    // The set of tentative nodes to be evaluated,
initially containing the start node
    came_from := the empty map    // The map of navigated nodes.




    g_score[start] := 0     // Cost from start along best known path.

    // Estimated total cost from start to goal through y.
    f_score[start] := g_score[start] + heuristic_cost_estimate(start, goal)

    while openset is not empty
        current := the node in openset having the lowest f_score[] valueif
        current = goal
            return reconstruct_path(came_from, goal)

        remove current from opensetadd
        current to closedset
        for each neighbor in neighbor_nodes(current)if
            neighbor in closedset
                continue
            tentative_g_score := g_score[current] +
dist_between(current,neighbor)

            if neighbor not in openset or tentative_g_score < g_score[neighbor]
                came_from[neighbor] := current
                g_score[neighbor] := tentative_g_score
                f_score[neighbor] := g_score[neighbor] +
heuristic_cost_estimate(neighbor, goal)
                if neighbor not in openset add
                    neighbor to openset

    return failure
```

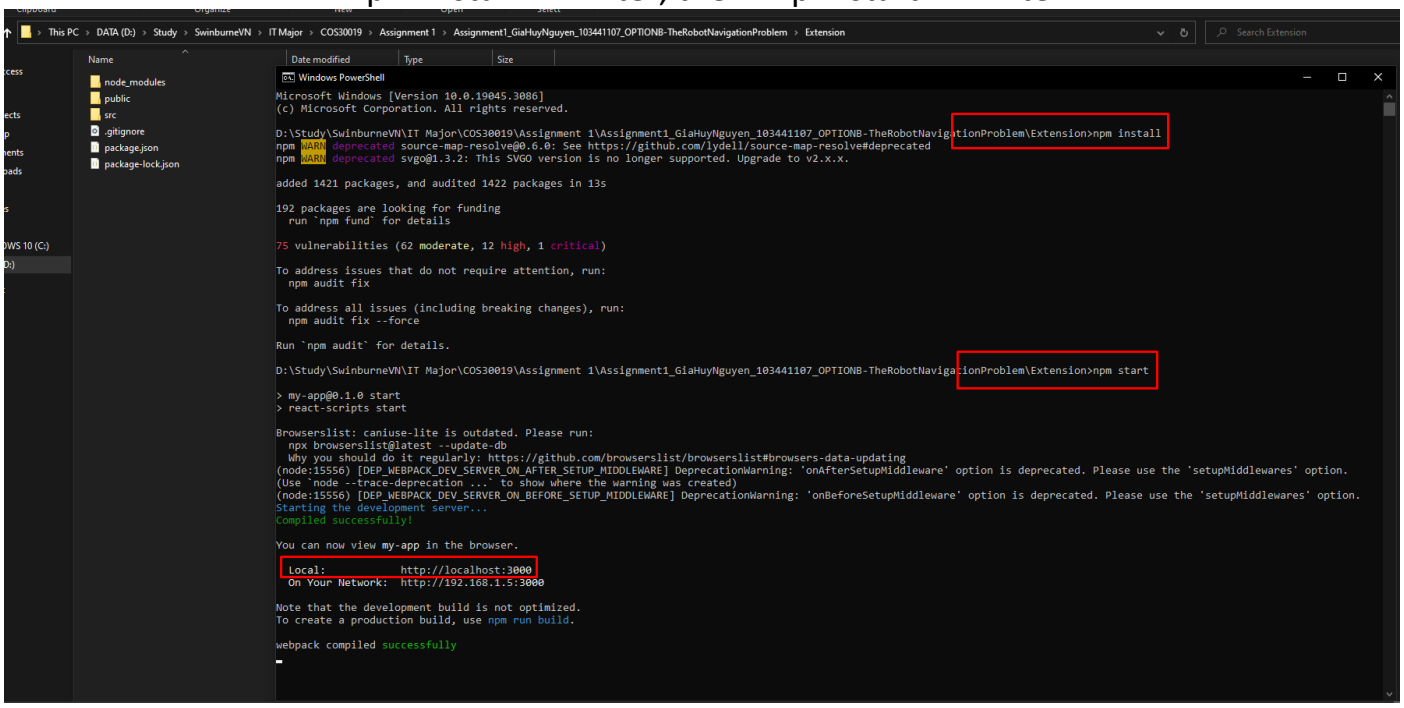
III. Implementation

I performed additional research into the graphical user interface (GUI) version of the pathfinding problem. To create a visualizer, I successfully converted the Python code into JavaScript. Fortunately, I was concurrently enrolled in the "COS30043-Interface Design and Development" course at Swinburne, which equipped me with valuable insights into JavaScript web development and frameworks. This knowledge proved instrumental in completing my research project. Furthermore, I invested time in watching YouTube tutorials and diligently following coding instructions to successfully accomplish the task. Regarding the GUI implementation, since I have deployed the interface using ReactJS, I will provide a concise guide on running the deployment on localhost.

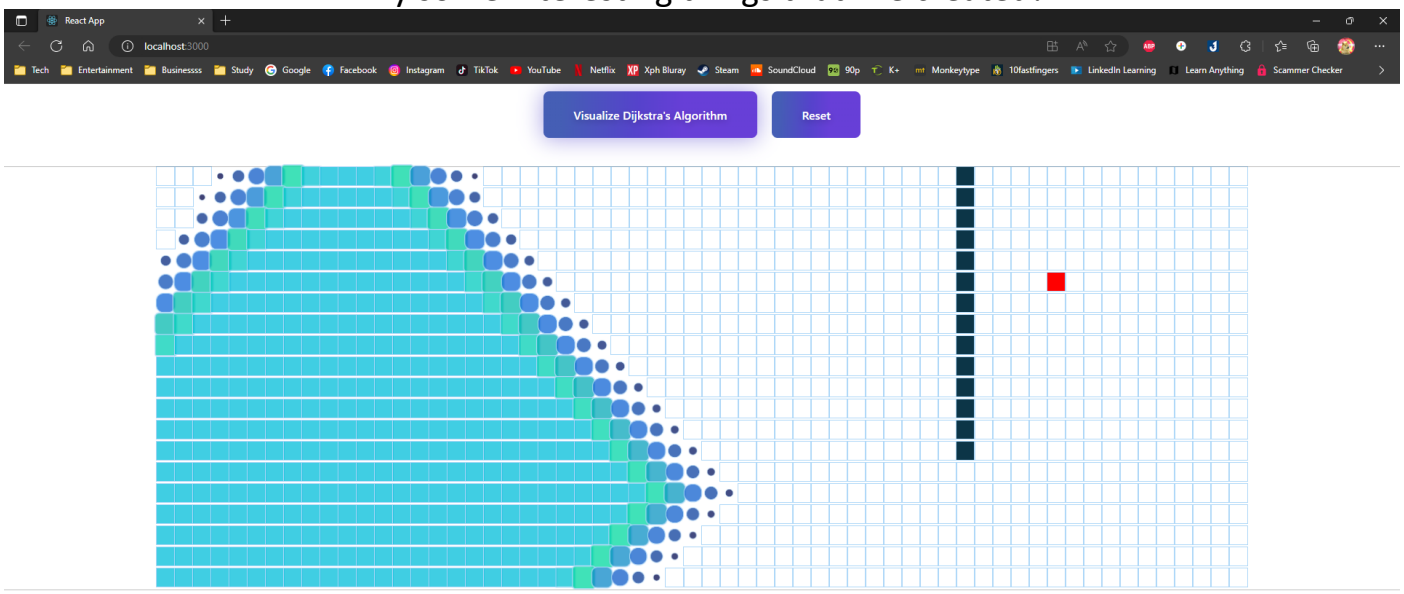
Firstly, go to the folder include the code project (Extension), on the search bar type: “cmd” then -> Enter.



When the Command Prompt (cmd) window opens, type the command: “npm install” -> Enter, then “npm start” -> Enter.



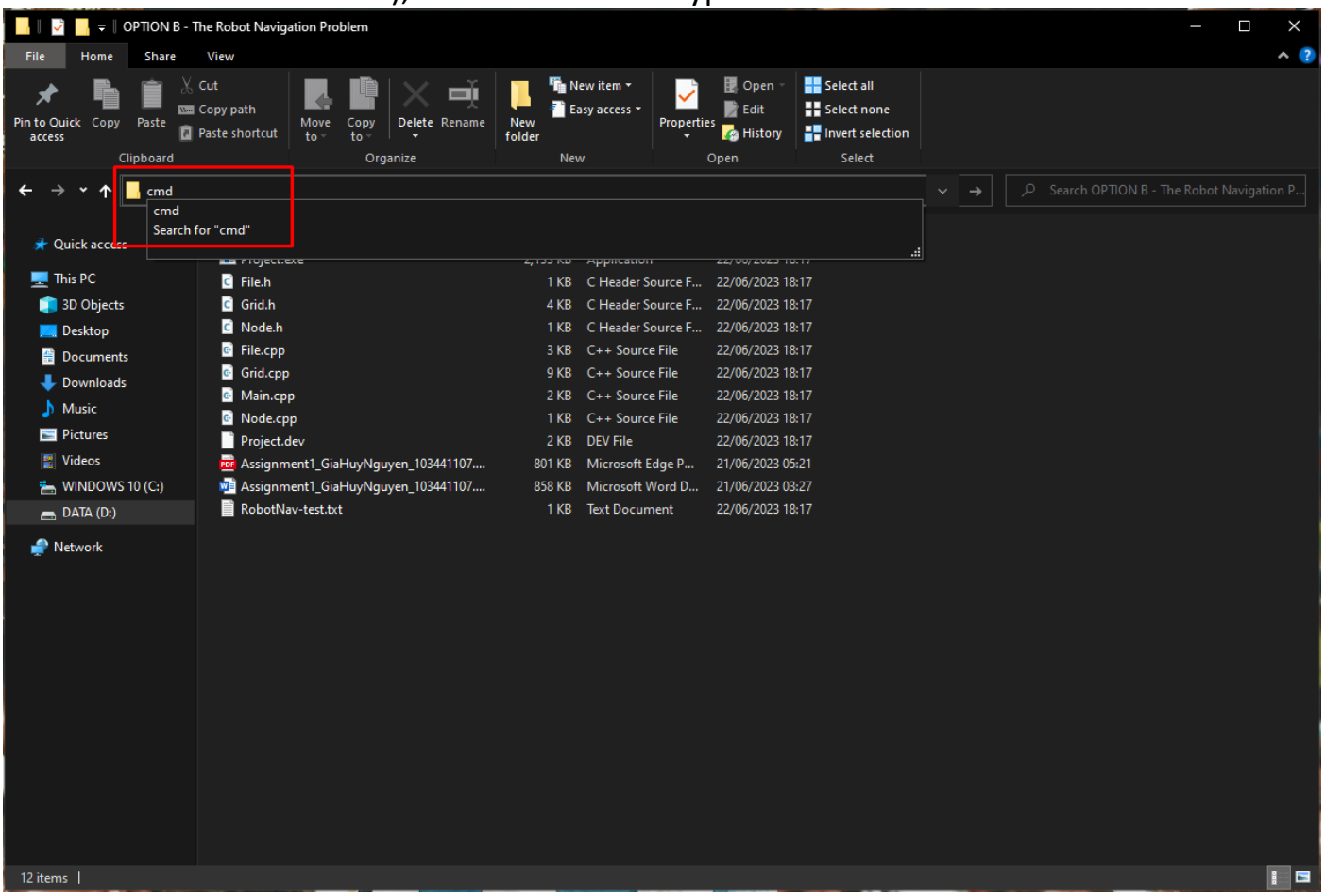
Now you can access my App in the browser by: http://localhost:3000
Try some interesting things that I've created!



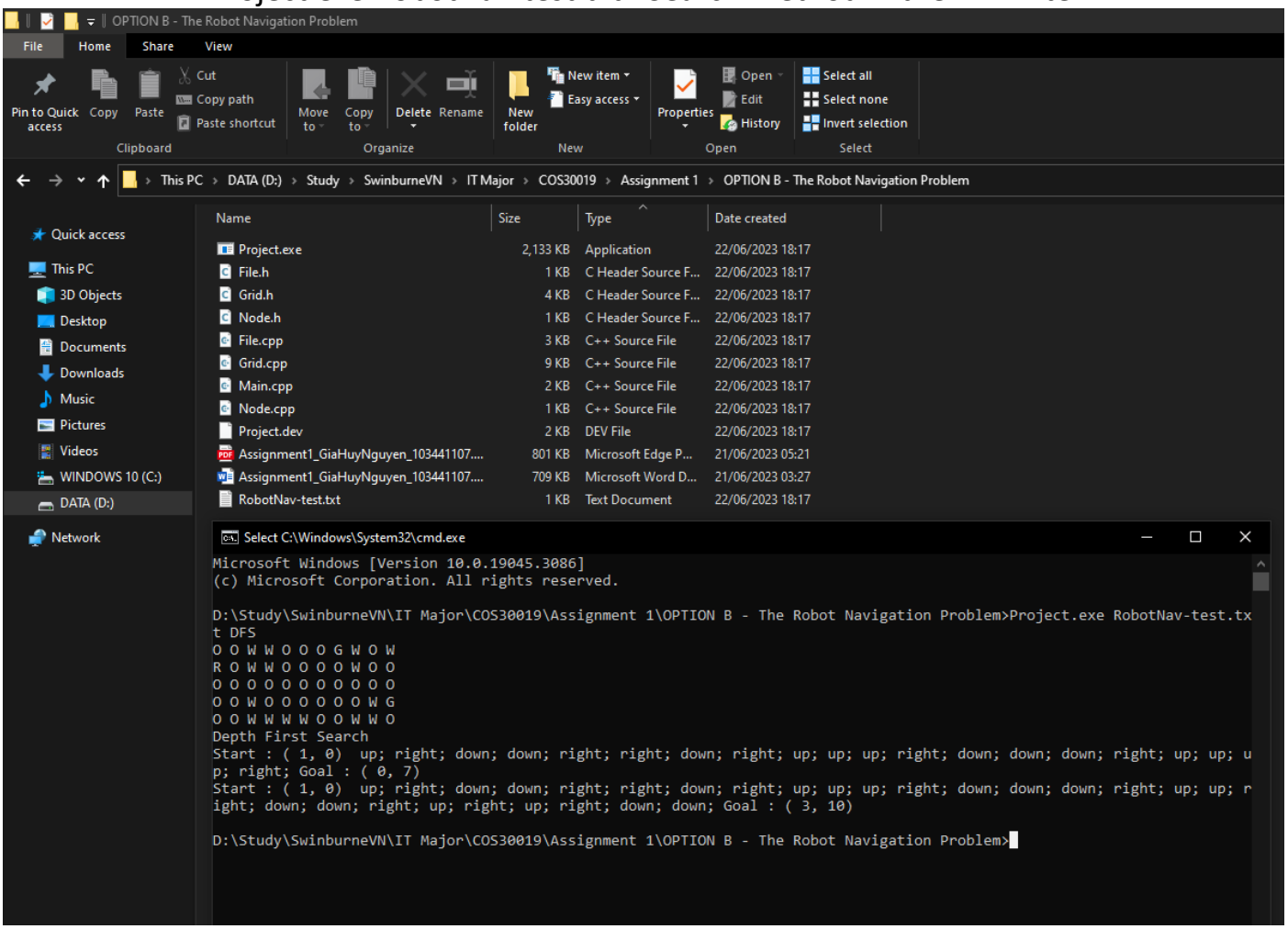
The brief tutorial demonstrates how to interact with the program. You can manually place walls/barriers between the default start and finish points. Once you've done that, simply click the "Visualize Dijkstra's Algorithm" button to begin the search. To restore the default settings, you can use the "Reset" button.

IV. How to run the program?

Firstly, go to the folder include the code project (OPTION B - The Robot Navigation Problem), on the search bar type: “cmd” then -> Enter.



When the Command Prompt (cmd) window opens, type the command:
”Project.exe RobotNav-test.txt <Search Method>” then -> Enter.



For example, I will have several commands like:
“ Project.exe RobotNav-test.txt **DFS** ”
“ Project.exe RobotNav-test.txt **BFS** ”
“ Project.exe RobotNav-test.txt **GBFS** ”
“ Project.exe RobotNav-test.txt **AS** ”
The result returned will be:
- Map where R is the initial position of the robot, G is the destinationpoint, W is the wall
- Search method name
- Search Results

V. Results & Conclusions

Based on the results that I have gathered below; I am able to draw the following conclusion:

1. DFS:

```
Select C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19045.3086]
(c) Microsoft Corporation. All rights reserved.

D:\Study\SwinburneVN\IT Major\COS30019\Assignment 1\OPTION B - The Robot Navigation Problem>Project.exe RobotNav-test.txt DFS
O O W W O O O G W O W
R O W W O O O O W O O
O O O O O O O O O O O
O O W O O O O O O W G
O O W W W W O O W W O
Depth First Search
Start : ( 1, 0)  up; right; down; down; right; right; down; right; up; up; up; right; down; down; down; right; up; up; up; right; Goal : ( 0, 7)
Start : ( 1, 0)  up; right; down; down; right; right; down; right; up; up; up; right; down; down; down; right; up; up; right; down; down; right; up; right; up; right; down; down; Goal : ( 3, 10)

D:\Study\SwinburneVN\IT Major\COS30019\Assignment 1\OPTION B - The Robot Navigation Problem>
```

Advantages: The algorithm requires less memory usage compared to other algorithms.
Disadvantages: The output paths generated by the algorithm can sometimes be less efficient compared to other algorithms.

2. BFS:

```
Select C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19045.3086]
(c) Microsoft Corporation. All rights reserved.

D:\Study\SwinburneVN\IT Major\COS30019\Assignment 1\OPTION B - The Robot Navigation Problem>Project.exe RobotNav-test.txt BFS
O O W W O O O G W O W
R O W W O O O O W O O
O O O O O O O O O O O
O O W O O O O O O W G
O O W W W W O O W W O
Breadth First Search
Start : ( 1, 0)  down; right; right; right; right; up; up; right; right; right; Goal : ( 0, 7)
Start : ( 1, 0)  down; right; right; right; right; right; right; right; right; right; right; down; Goal : ( 3, 10)

D:\Study\SwinburneVN\IT Major\COS30019\Assignment 1\OPTION B - The Robot Navigation Problem>
```

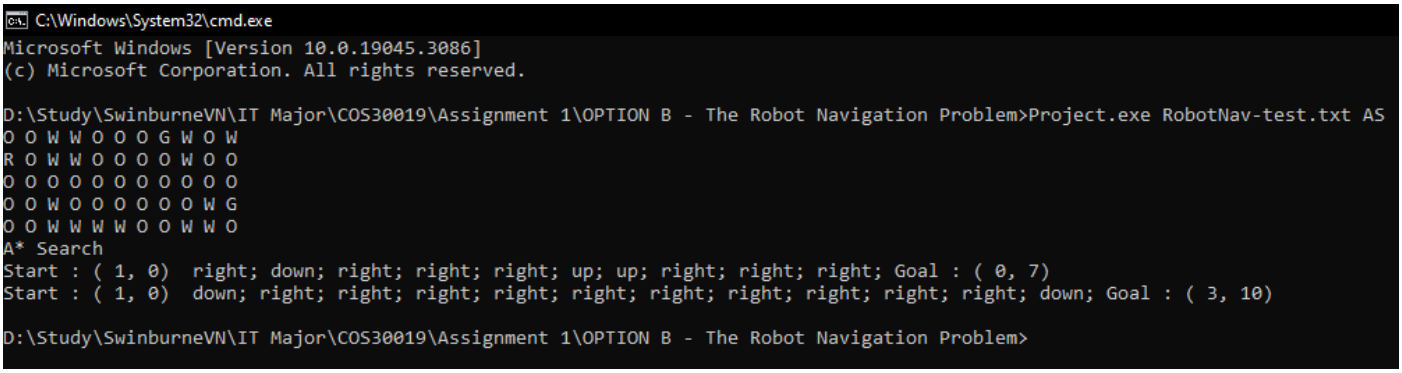
Advantage: It is more efficient than Depth-First Search (DFS) in certain cases.
Disadvantage: It requires a significant amount of memory.

3. GBFS:

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19045.3086]
(c) Microsoft Corporation. All rights reserved.

D:\Study\SwinburneVN\IT Major\COS30019\Assignment 1\OPTION B - The Robot Navigation Problem>Project.exe RobotNav-test.txt GBFS
O O W W O O O G W O W
R O W W O O O O W O O
O O O O O O O O O O O
O O W O O O O O O W G
O O W W W W O O W W O
Greedy Best First Search
Start : ( 1, 0)  right; down; right; right; right; up; up; right; right; right; Goal : ( 0, 7)
Start : ( 1, 0)  down; right; right; right; down; right; right; right; right; right; up; right; right; down; Goal : ( 3, 10)

D:\Study\SwinburneVN\IT Major\COS30019\Assignment 1\OPTION B - The Robot Navigation Problem>
```

Advantage: It offers better efficiency compared to Breadth-First Search (BFS) and DFS, while utilizing the same amount of memory as BFS.
Disadvantages: The smallest cost value may not always result in the most efficient solution.

4. A*:



Advantages: It provides highly efficient paths.

***)** These conclusions highlight the trade-offs and strengths of each algorithm, which can help in choosing the most suitable one for specific scenarios.

=> In the Robot Navigation Problem, my preference is to employ the A* Search Method Algorithm.

VI. References:

1. Wikipedia for the definition as well as a deep explanation of thealgorithms: Wikipedia

2. This GitHub directory is for the Custom Search method though I findthese interesting, I cannot implement them yet :

Shortest Path (fribbels.github.io)

3. General Idea of the Robot Navigation Program:

Conceptual Bases of Robot Navigation Modeling, Control and Applications | IntechOpen

Full article: A comprehensive study for robot navigation techniques (tandfonline.com)

Bài 22. Tìm đường đi trong mê cung (part 1), đường đi robot. - YouTube

Python Path Finding Tutorial - Breadth First Search Algorithm - YouTube

A* Pathfinding Visualization Tutorial - Python A* Path Finding Tutorial - YouTube

4. Dijkstra Algorithm (research):

Python Dijkstra Algorithm – Be on the Right Side of Change (finxter.com)

How Dijkstra's Algorithm Works - YouTube

Dijkstra's Algorithm - Computerphile - YouTube

Dijkstra's Algorithm - YouTube

5. Implementation (research and tutorial):

YouTube video(especially Pathfinding Visualizer Tutorial (software engineering project) - YouTube,) and discussions with friends were immensely helpful to me. I also utilized various resources(especially How to Create a Path-Finding Algorithm Visualizer with React (freecodecamp.org)) to gain a deeper understanding of algorithms, with better illustrations and graphic examples. In certain instances, I found tutorials necessary to watch, such as those on batch file creation or CSS code in the visualizer, to successfully complete my program.