INFORMATICS INSTITUTE OF TECHNOLOGY

In Collaboration with

UNIVERSITY OF WESTMINSTER

Module Name **- ALGORITHMS**

Module Code – **5SENG003C.2**

Module Leader – **Mr. SIVARAMAN RAGU**

Written by

Mr. Dilshan Zarook | 20220088 | w1953568

2024

5. a). Choice of Algorithm and Data structures

Algorithm

Several algorithms are available for pathfinding, including Dijkstra's, Breadth-First Search, Depth-First Search, Greedy Best-First Search, and others. Based on the given scenario, I have selected A* (A-Star) to find the shortest path. The maze in this scenario has a start and end position and is obstructed by walls and rocks. This algorithm, which is most frequently utilized in games, ranks the paths that are most likely to lead to the goal by combining it with the Dijkstra heuristic. I have selected this algorithm over others for a few reasons.

Efficiency is one factor; for large mazes, this is more efficient than Dijkstra's. The primary explanation is that it estimates the cost of achieving the objective from a given node using a heuristic. As it prioritizes the nodes, this will always result in fewer nodes that need to be investigated.

Flexibility is another key factor; it can manage a variety of challenges, including the tendency to slide. We can assume that traveling through a rock will cost more than traveling through open areas. The heuristic function will never overestimate the cost to reach the goal, and the estimated cost from the current node to the goal is not greater than the cost from the current node to a neighbor plus the estimated cost from the neighbor to the goal, so using A* will always result in the shortest path.

Data Structures

I have used PriorityQueue(open sets). It keeps track of Node objects, which stand in for possible search locations to investigate. A heuristic estimate of the remaining distance to the objective is combined with the g-score (moving cost thus far) to determine the f-score, which determines the priority of nodes. This guarantees that the most promising avenues are investigated first in the search.

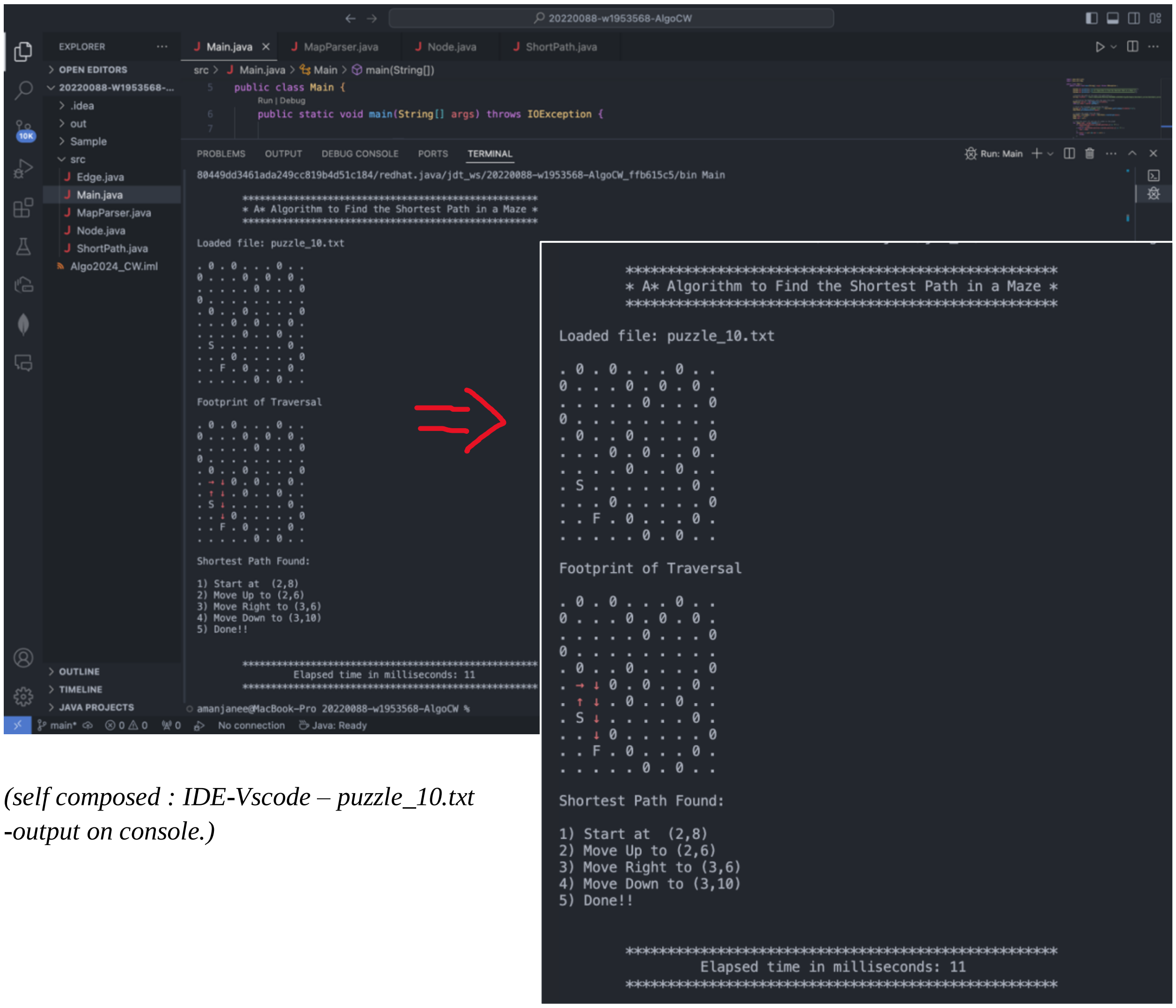
Custom Node class has been created to store relevant information such as x,y coordinates,fScore and gScore.

Maps (gScore, fScore, cameFrom): These are all instances of the java.util.HashMap class. cameFrom: Every examined point in the maze's parent node is tracked by this map. It's essential for rebuilding the ultimate route when the objective is accomplished. The parent Node object is the value, and the node's coordinates (x,y) are represented as a string in the key. gScore: The movement cost, or g-score, for every examined position is stored on this map. Once more, the value is the computed g-score, and the key is the string representation of the coordinates.

fScore: For every examined position, the f-score (total estimated cost) is stored on this map. It is computed by multiplying the heuristic value derived from the heuristic function by the g-score from gScore.

ArrayList used to read the maze from the file and store. char[][] (map): The representation of the labyrinth is stored in this 2D character array, where each character stands for a distinct element (e.g., 'S' for start, 'F' for finish, '0' for obstacle, and empty spaces for paths that can$ be traversed).

b). A run of my algorithm on puzzle_10.txt in the benchmark series



```
80449dd3461ada249cc819b4d51c184/redhat.java/jdt_ws/20220088-w1953568-AlgoCW_ffb615c5/bin Main

        ****************************************************
        * A* Algorithm to Find the Shortest Path in a Maze *
        ****************************************************

Loaded file: puzzle_10.txt

. 0 . 0 . . . 0 . .
0 . . . 0 . 0 . 0 .
. . . . . 0 . . . 0
0 . . . . . . . . .
. 0 . . 0 . . . . 0
. . . 0 . 0 . . 0 .
. . . . 0 . . 0 . .
. S . . . . . . 0 .
. . . 0 . . . . . 0
. . F . 0 . . . 0 .
. . . . . 0 . 0 . .

Footprint of Traversal

. 0 . 0 . . . 0 . .
0 . . . 0 . 0 . 0 .
. . . . . 0 . . . 0
0 . . . . . . . . .
. 0 . . 0 . . . . 0
. → ↓ 0 . 0 . . 0 .
. ↑ ↓ . 0 . . 0 . .
. S ↓ . . . . . 0 .
. . ↓ 0 . . . . . 0
. . F . 0 . . . 0 .
. . . . . 0 . 0 . .

Shortest Path Found:

1) Start at  (2,8)
2) Move Up to (2,6)
3) Move Right to (3,6)
4) Move Down to (3,10)
5) Done!!


        ****************************************************
              Elapsed time in milliseconds: 11
        ****************************************************
amanjanee@MacBook-Pro 20220088-w1953568-AlgoCW %
```

(self composed : IDE-Vscode – puzzle_10.txt -output on console.)

c). Performance Analysis of the algorithm and implementation

Puzzle Name	**Input Size**	**Time (Average) – Milliseconds Approximately**
Puzzle_10	**10*10**	**11**
Puzzle_20	**20*20**	**12**
Puzzle_40	**40*40**	**15**
Puzzle_80	**80*80**	**25**
Puzzle_160	**160*160**	**33**
Puzzle_320	**320*320**	**51**
Puzzle_640	**640*640**	**65**
Puzzle_1280	**1280*1280**	**103**
Puzzle_2560	**2560*2560**	**220**

(self-composed Table of Time complexity for input and run time of A algorithm)*

The **Empirical method** approach provides a practical insight into an algorithm real world performance and helps in optimizing it fir efficiency and scalability. The input size is grown by factor of 2 (or doubling) the runtime of the algorithm is grown approximately by a factor of

12 /11 = 1.0
15/12 = 1.25
25 /15 = 1.6 => 2
33/25 = 1.32
51/33 = 1.5 => 2
65/51 = 1.27
103/65 = 1.5 => 2
220/103 = 2.2. => 2

As a result of my algorithm this is belongs to Linear complexity class and big O notation is O(n). hence it proved that algorithm used heuristic method efficiency.