****

KULLIYYAH OF INFORMATION AND COMMUNICATION TECHNOLOGY

CSC 2104 | Section 3

REPORT

**Maze Solving**

Group Name: Golden Sky

Group Member:

|  |  |
| --- | --- |
| Mutiullah Samim | 1325481 |
| Nasratullah Habib | 1324281 |
| Mohamad Ullah Asrar | 1210871 |
| Zulfani Ahmad Zuhri | 1420625 |

Lecturer: Dr. Imad Muhsin Thaha

Tutor: Bro Abdullah Ahmad Zarir

**Maze Solving**

Mutiullah Samim, Nasratullah Habib,

Mohamad Ullah Asrar, Zulfani Ahmad Zuhri

*Kulliyah of Information & Communication Technology*

*International Islamic University Malaysia*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Abstract** - This report is made to show the explanation detail of maze solver algorithm, its functions/classes, and extra features added, and maze generator scheme.

Keywords – Maze, A\*, DFS, BFS.

1. **Introduction**

The program will show random maze generator with three options of maze solver algorithms included to solve the maze generated by the user which are A\* search algorithm, Depth First Search (DFS) algorithm, and Breadth First Search algorithm. These maze solvers will find a possible route from starting point to the exit point. There are some additional features in our program which are maze generator button, and clear maze button.

1. **Algorithms**

We have three algorithms (DFS, BFS and A\* Search).

The DFS algorithm are designed to be used inside the maze by a traveler with no prior knowledge of the maze, whereas the other two algorithms, A\* and BFS, are designed to be used by a person or computer program that can see the whole maze at once.

* 1. Depth First Search (DFS) algorithm

To implement DFS using recursive method or stack. Basically it start from random point and keep digging path in one of 4 directions (up, right, down, left) and choose the deepest way until it can’t go any further. Once it stuck, then take a step back until it find an open path. And would continue digging from there. It’s just the repetition of these.

* 1. Breadth First Search (BFS) algorithm

Breadth-first algorithm works with the method branching from the starting cell to the neighbor cells (just traversable cells), (untraversable cells and cells out of boundaries are discarded) until the goal cell is found. Every traversable neighbor cell is added to an array which is called OPEN LIST. OPEN LIST is the array of neighbor cells which must be reviewed in order to find the goal cell.

Or is an [algorithm](https://en.wikipedia.org/wiki/Algorithm) for traversing or searching [tree](https://en.wikipedia.org/wiki/Tree_data_structure) or [graph](https://en.wikipedia.org/wiki/Graph_(data_structure)) data structures. It starts at the [tree root](https://en.wikipedia.org/wiki/Tree_(data_structure)#Terminology)or starting point) and explores the neighbor nodes first, before moving to the next level neighbors.

* 1. A\* search algorithm

A\* is an [informed search algorithm](https://en.wikipedia.org/wiki/Informed_search_algorithm), or a [best-first search](https://en.wikipedia.org/wiki/Best-first_search), meaning that it solves problems by searching among all possible paths to the solution (goal) for the one that incurs the smallest cost (least distance travelled, shortest time, etc.), and among these paths it first considers the ones that *appear* to lead most quickly to the solution. It is formulated in terms of [weighted graphs](https://en.wikipedia.org/wiki/Weighted_graph): starting from a specific [node](https://en.wikipedia.org/wiki/Node_(graph_theory)) of a graph, it constructs a [tree](https://en.wikipedia.org/wiki/Tree_(data_structure)) of paths starting from that node, expanding paths one step at a time, until one of its paths ends at the predetermined goal node.At each iteration of its main loop, A\* needs to determine which of its partial paths to expand into one or more longer paths. It does so based on an estimate of the cost (total weight) still to go to the goal node. A\* search and BFS are almost same but A\* search is using linked list in the implementation rather than array.

1. **Major Functions/ Classes**

* Public class myMaze: the main form of the program. Located in the center of the screen
* Public static class MazePanel extends Jpanel: This class defines the contents of the main form and contains all the functionality of the program.
* Private class Cell: Cell class that represents the cell of the grid
* Private class ActionHandler: When the user presses a button performs the corresponding functionality
* Private class RepaintAction: The class that is responsible for the animation
* Private class MyMaze: Class that creates a random maze
* Void addNeighbor(Cell other): add a neighbor to this cell, and this cell as a neighbor to the other
* Private Void Mazebuttonactionperformed(Java.Awt.Event.Actionevent Evt: Function Executed If The User Presses The Button "Maze"
* Private Void Initializegrid(Boolean Makemaze): Creates A New Clean Grid Or A New Maze
* Private Void Generatemaze(Int X, Int Y): Generate The Maze From Coordinates X, Y
* Public Cell Getcell(Int X, Int Y): Used To Get A Cell At X, Y; Returns Null Out Of Bounds
* JRadioButton dfs, bfs, aStar: buttons for selecting the algorithm
* Boolean Found: Flag That The Goal Was Found
* Boolean Searching: Flag That The Search Is In Progress
* Boolean Endofsearch: Flag That The Search Came To An End
* Int Delay: Time Delay Of Animation (In Msec)
* Repaintaction Action = New Repaintaction(): The Object That Controls The Animation
* Timer timer: the Timer which governs the execution speed of the animation
* ButtonGroup algoGroup = new ButtonGroup(): ButtonGroup that synchronizes the three RadioButtons choosing the algorithm, so that only one can be selected anytime
* Private void expandNode(): Expands a node and creates his successors
* Private Arraylist<Cell>: Creates The Successors Of A State/Cell
* Private Void Plotroute(): Calculates The Path From The Target To The Initial Position Of The Robot, Counts The Corresponding Step And Measures The Distance Traveled.
* Private Void Fillgrid(): Gives Initial Values For The Cells In The Grid. With The First Click On Button 'Clear' Clears The Data Of Any Search Was Performed (Frontier, Closed Set, Route) And Leaves Intact The Obstacles And The Robot And Target Positions In Order To Be Able To Run Another Algorithm With The Same Data. With The Second Click Removes Any Obstacles Also.
* Public Void Paintcomponent(Graphics G): Fills The Background Color.

1. **Extra features added**

* Maze generator button: This feature enable user to use previous maze for other algorithms or other use with same maze scheme rather than it is automated change randomly. When user wants to use different maze scheme he just push button maze and then the maze scheme will be generated randomly.
* Clear button: This feature will enable user clear or clean the maze scheme and use the previous scheme for other use. It also enable user to stop the program when the program running to change or quit the maze.

1. **Maze generator scheme**

MyMaze: the maze is random whenever you click maze button the maze will generate randomly every time and the maze represents by the grid. Each cell in grid represents a path or wall. In the maze we have walls and paths when the solver algorithm solve the maze it will look for open path if there is wall it will check the other neighbor to find the another open path and solve it according to the algorithm to reach to the end point and the found path from starting point to the target point in a maze is represent by yellow color.