University of Science and Technology Chittagong

Name : Prantika Nath

ID : 22010106

Batch : 38th

Dept : CSE

**Explanation of DFS code:**

Depth-First Search (DFS) is a fundamental algorithm used for traversing or searching through graph and tree data structures. It starts at a selected node (the root in a tree or an arbitrary node in a graph) and explores as far as possible along each branch before backtracking. This means it goes deep into one branch of the graph until it reaches the end, then it backtracks to explore other branches.

**How the code work:**

The code defines a Maze class that reads a maze from a text file, sets up the maze structure, and implements a DFS algorithm to find a path from a starting point (A) to a goal point (B). It also includes functionality to visualize the maze and the solution.

In this code :

Class note refers each position in the maze with its state (coordinates), parent (previous node), and the action taken to reach it.

Frontier Classes refers StackFrontier and QueueFrontier. StackFrontier: Implements a stack for DFS. Nodes are added to the end and removed from the end (last in, first out) and QueueFrontier Can be used for BFS but is not utilized in this code.

The Maze Class is the main class of this code. The part work with

Initialization (\_\_init\_\_) , Print Method, Neighbors Method, Solve Method, Output Image Method.

1. In initialization(\_\_init\_\_) Reads the maze from a file, determines its dimensions, and identifies walls, the start point (A), and the goal (B).
2. In Printmethod Displays the maze in the console, marking walls, the start, the goal, and the solution path.
3. In , Neighbors Method returns valid neighboring positions (up, down, left, right) for a given state, ensuring they are within bounds and not walls.
4. In Solve Method implements the DFS algorithm to find a path from start to goal. Initializes the frontier with the start node and an explored set. Repeatedly removes a node from the frontier, checks if it’s the goal, and adds its neighbors to the frontier if they haven’t been explored.
5. In **Output Image Method** Visualizes the maze and solution using the PIL library, creating a PNG image of the maze.

The another part in this code is **Execution.**

The Execution part work like:

The code checks for command-line arguments to get the maze file name.

It creates a Maze object, prints the initial maze, solves it, and then prints the number of explored states and the solution. Finally, it saves an image of the maze with the solution path highlighted.

Run the code:

For run the code we have to install a Python module called : “**pillow” .**  For download the module the command is **: pip install pillow**.

And finally in output screen we have to give the command called : **python maze.py maze.txt**

**Output:**

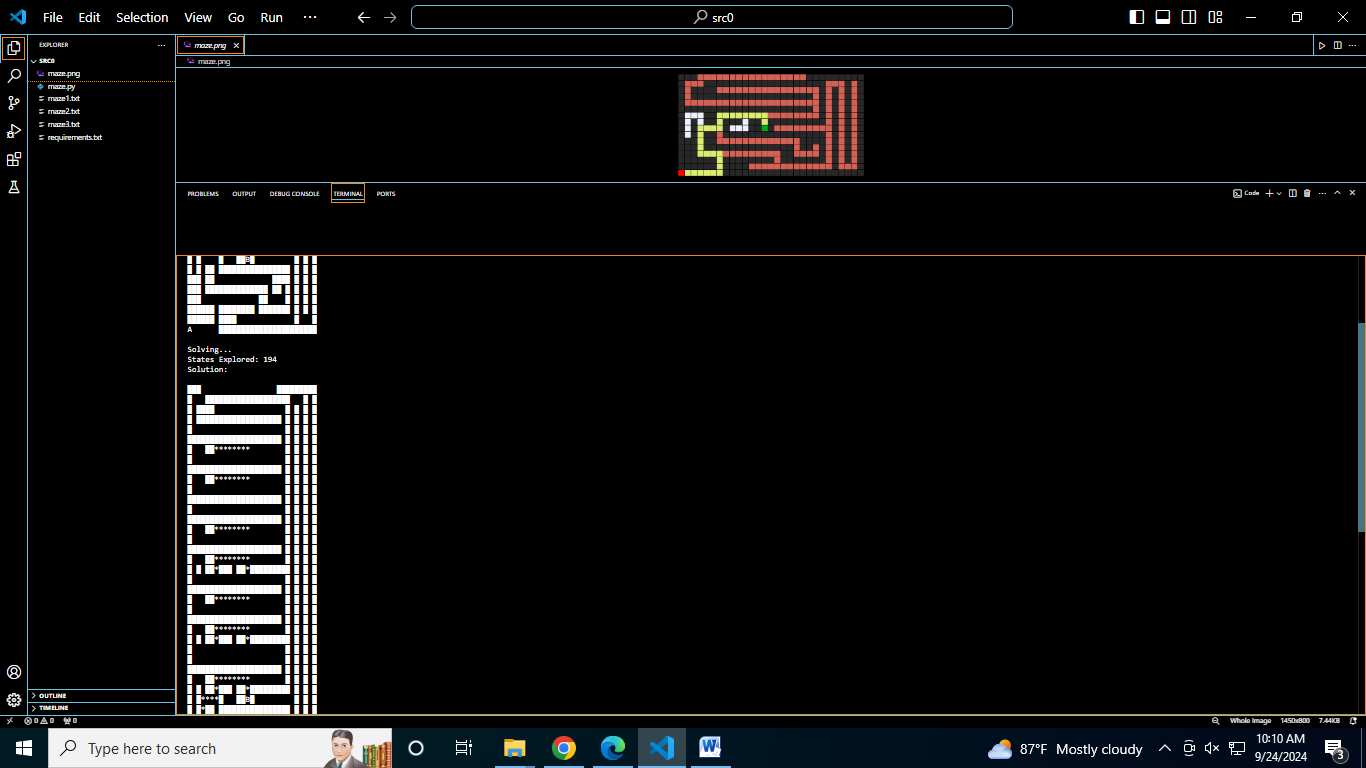


Fig: maze2.txt

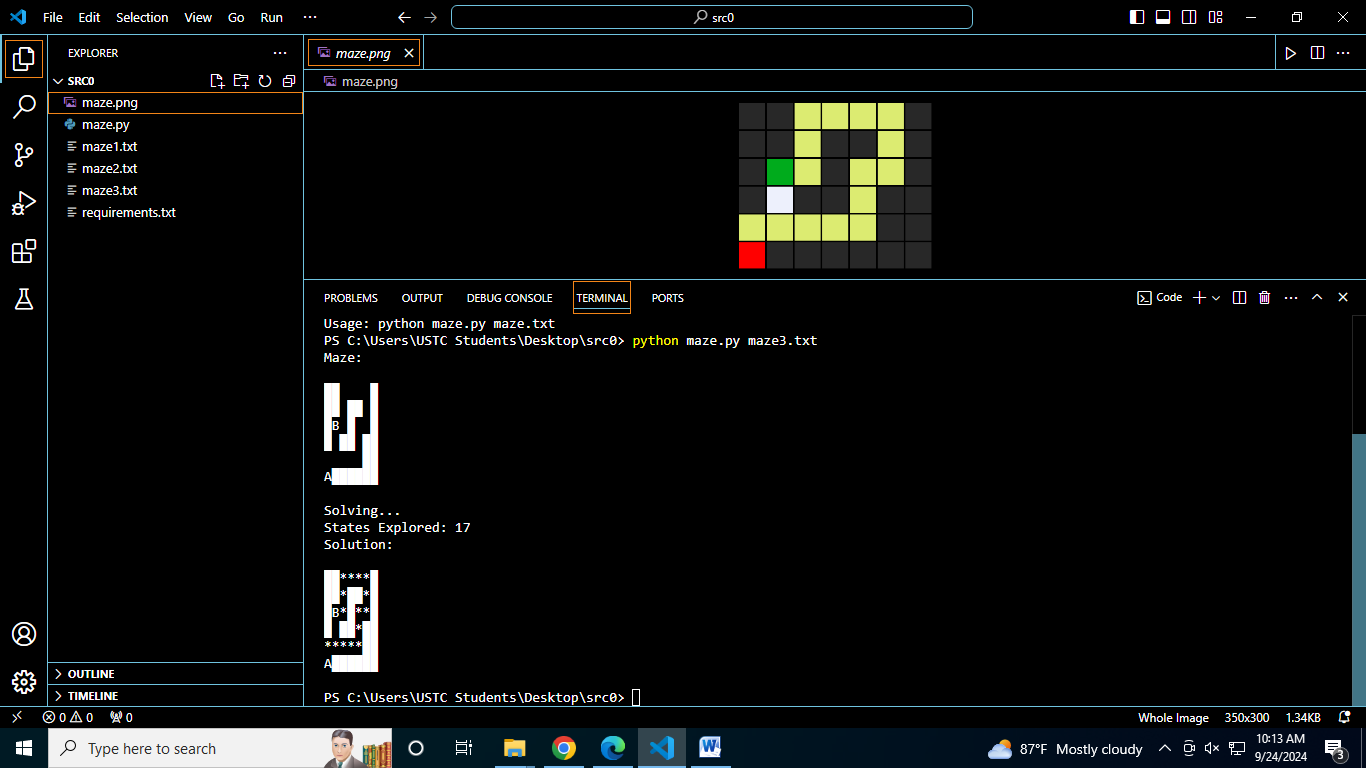


Fig: maze3.txt