MAZE GENERATOR AND SOLVER

PRESENTED BY

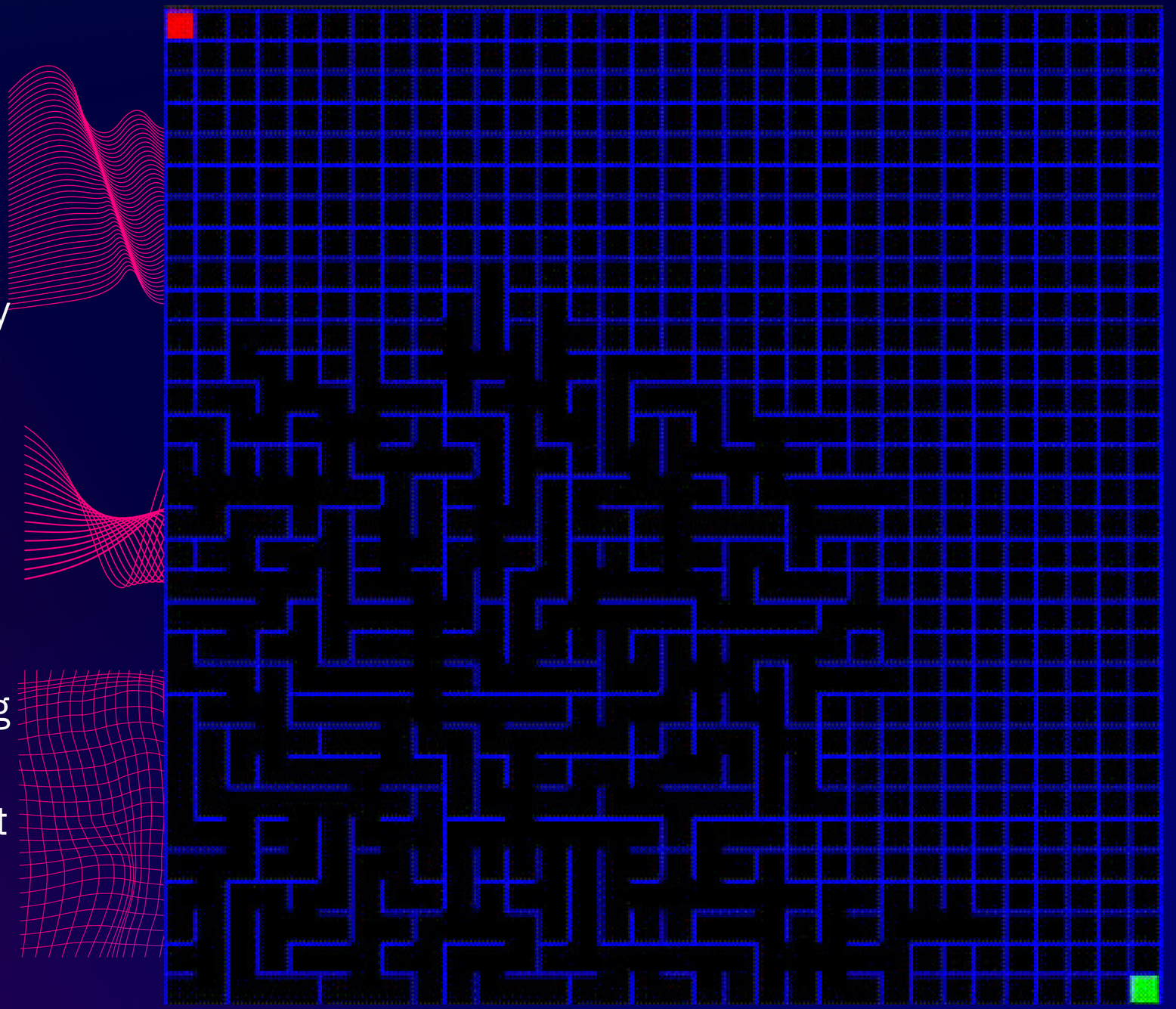
Shrutya Chawla(2022487)

Vanshika Pal(2022560)

Nandini Jain(2022316)

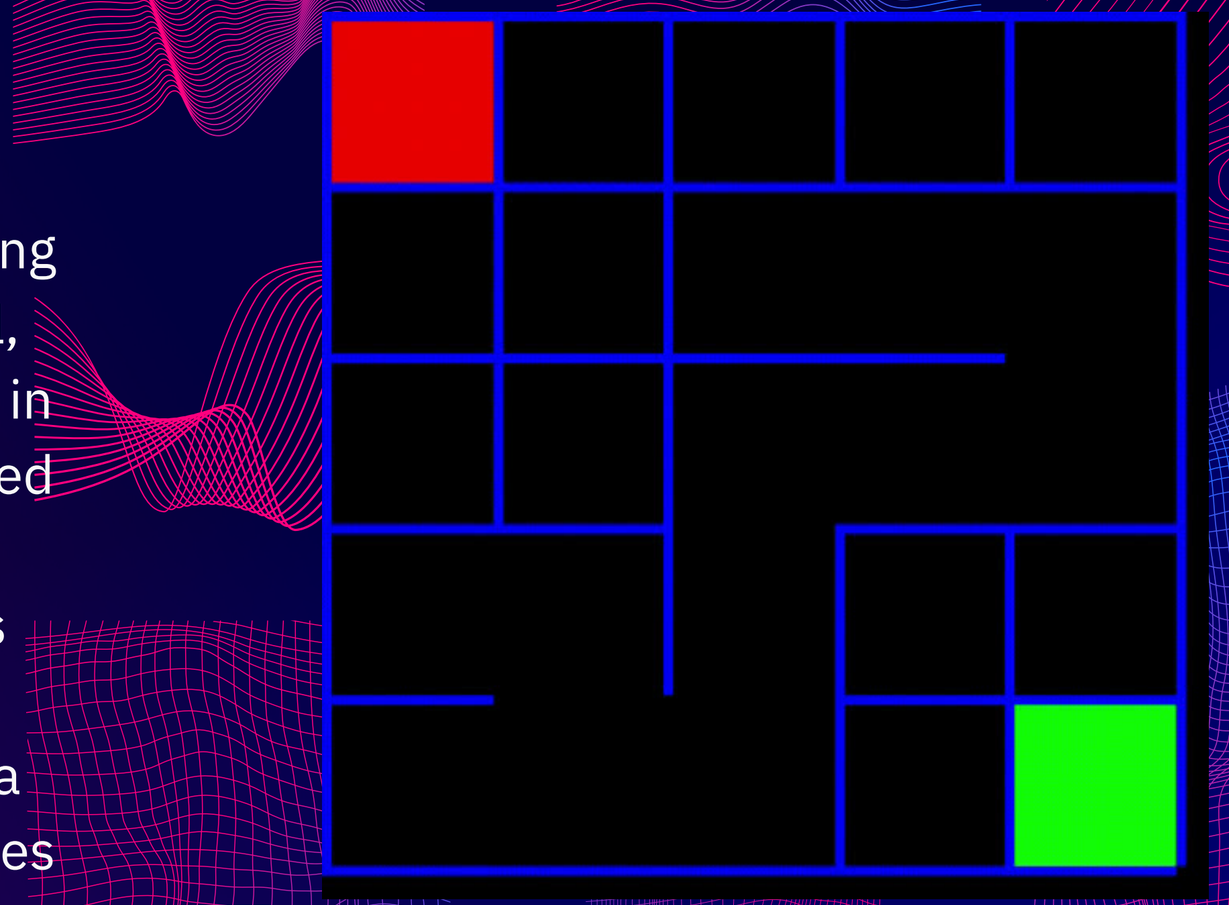
Prims Algorithm

Prim's Algorithm generates mazes by randomly selecting walls and carving passages between cells, ensuring a fully connected maze. It starts with a grid of cells, each representing a maze part. A set tracks visited cells, and a list processes them. Starting at a random cell, the algorithm adds neighboring walls to the list, selects a random wall, and carves a passage to the neighboring cell. This repeats until the list is empty, creating a fully connected maze without loops.



Binary Tree Algorithm

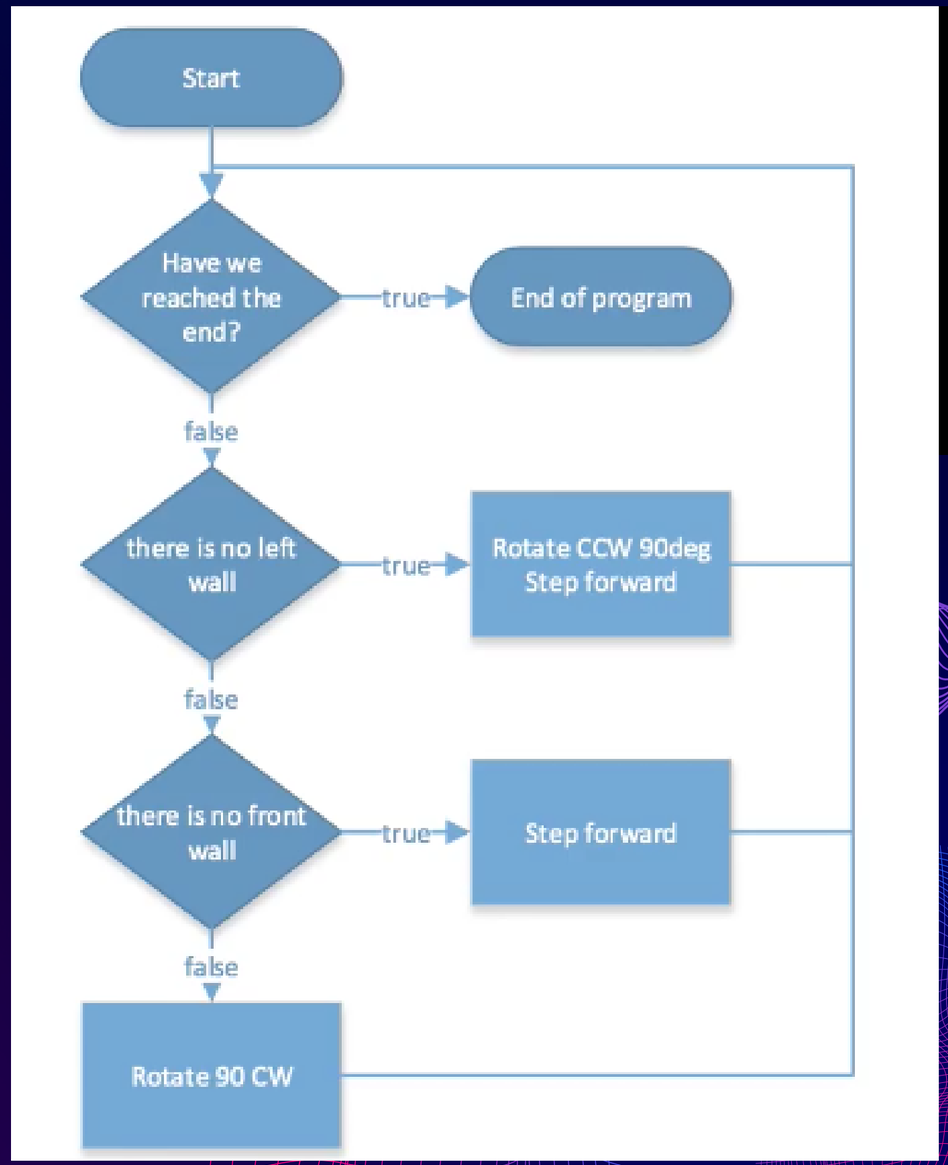
Binary Tree Maze Generation creates mazes using a simple process. The algorithm operates on a cell grid, initializing each cell with information about paths and visitation status. Starting randomly, it iterates through each cell, choosing a random direction. If moving in that direction is valid, a passage is carved between the current cell and the neighboring one. Backtracking occurs when a cell lacks valid neighbors, ensuring all cells are visited, creating a fully connected maze. The algorithm uses a stack to track the current cell, and the grid maintains information about paths and visited cells.

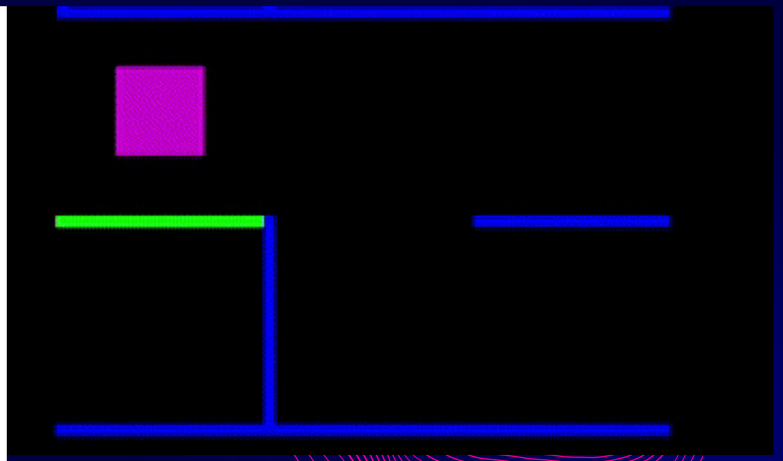


Left Wall Follower

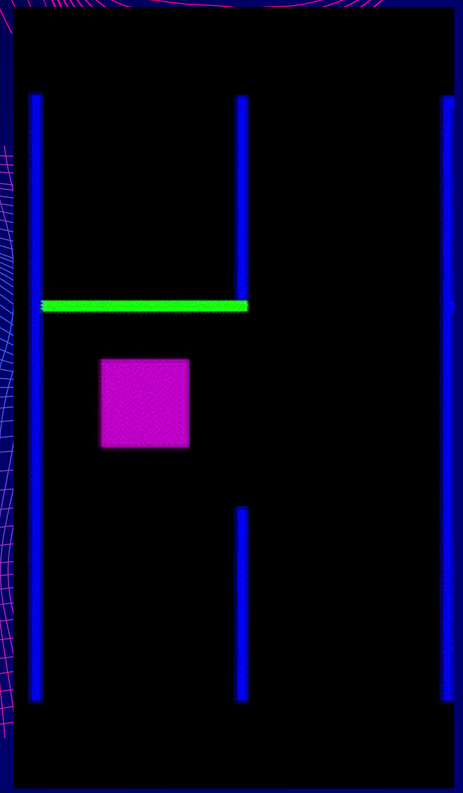
The algorithm always follows the left wall of the maze

The block rotates clockwise or anticlockwise depending on the placement of the walls around the agent





Anti-Clockwise

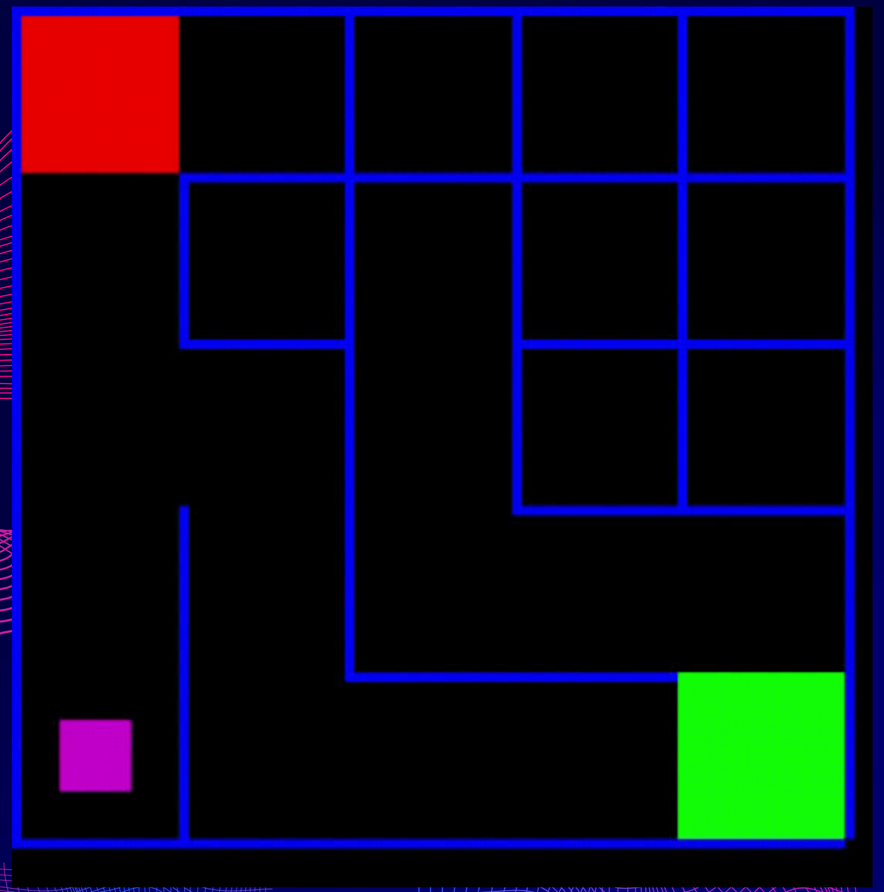


Clockwise

Dead-End Filler

The script iterates through the maze cells, identifying and filling dead-end cells until no more dead-end cells remain. Note that more dead ends will be produced after each iteration of the algorithm, which will be consequently filled again.

The algorithm then simply traverses the only path left in the maze.



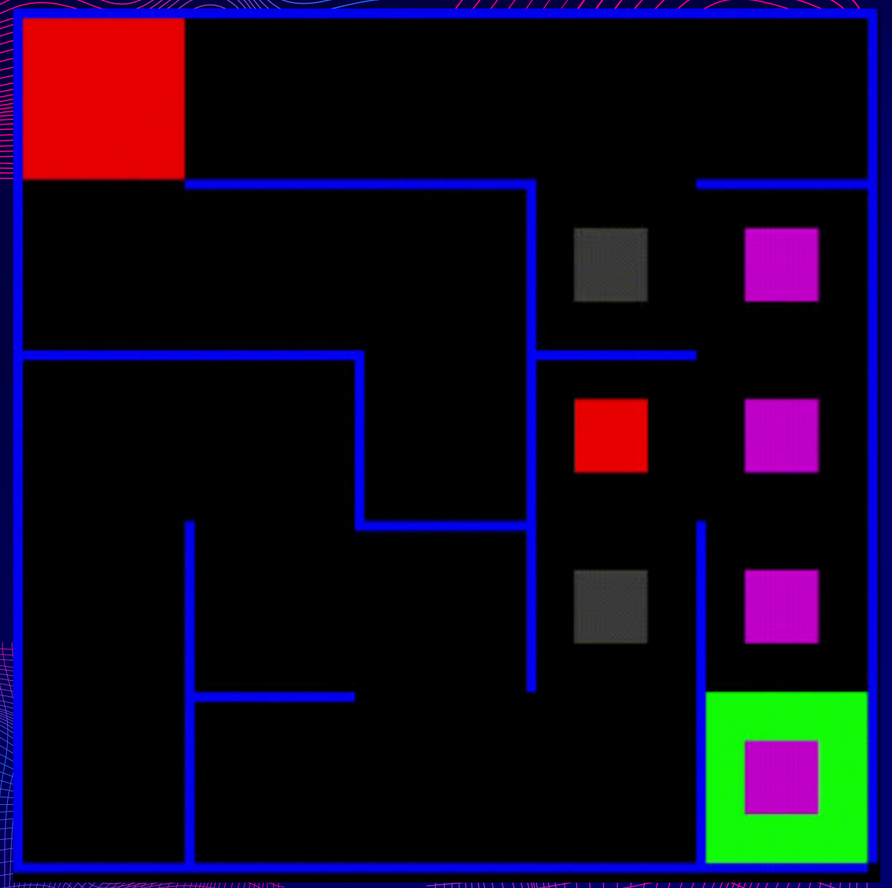
Dijkstra Algorithm

The algorithm explores the maze by visiting neighboring cells and updating the minimum distance to each cell.

The algorithm sets up a parent-child relationship between the cells in the dictionary. A cell is added to the dictionary only if the distance from the parent cell is less than previous weight of the child cell.

The Red color shows the current cell being explored and the grey color shows the neighbors of the current cell and the pink color shows the visited cells.

Once the goal is reached the algorithm backtracks the dictionary and the path is shown .

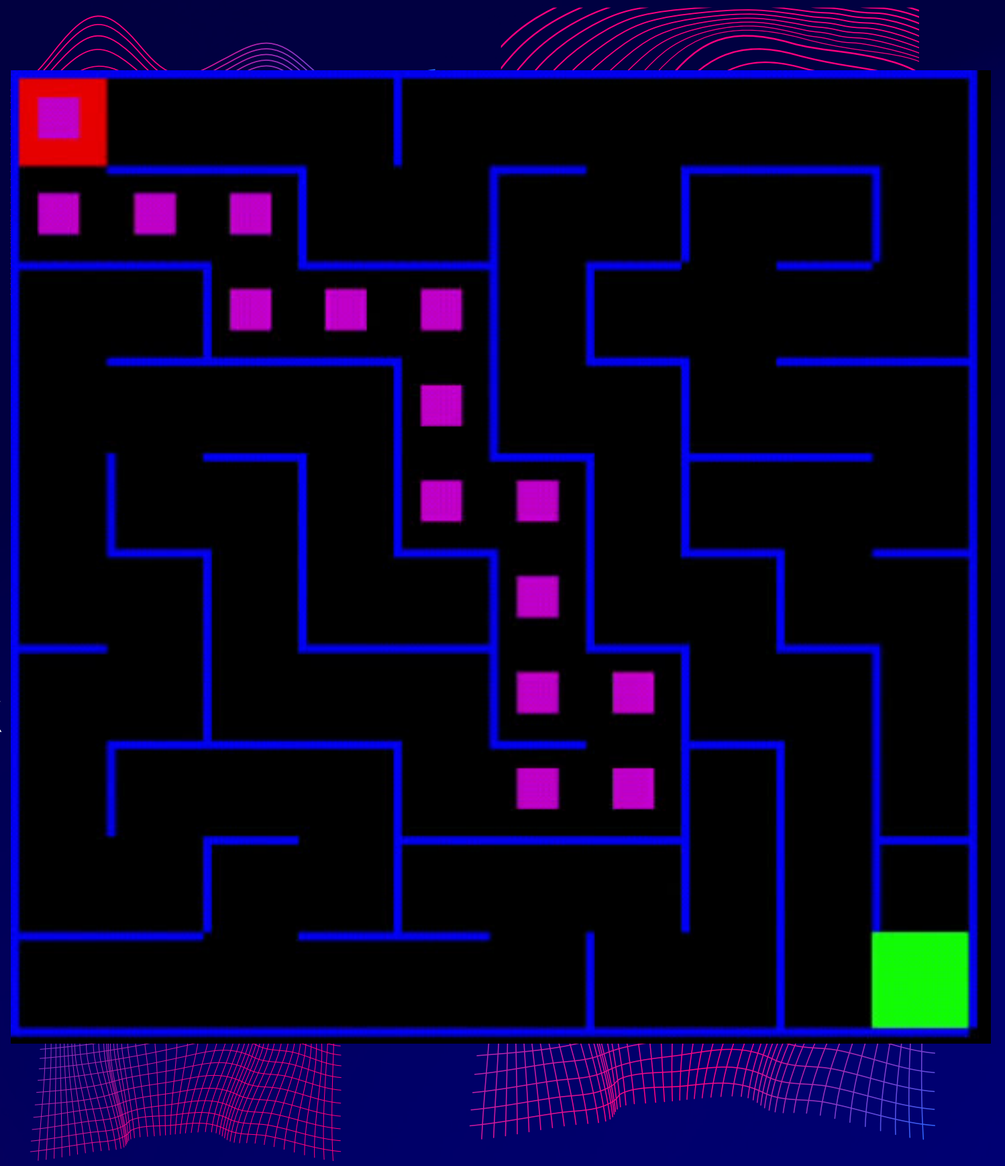


DFS Algorithm

A stack is kept to keep track of the current cells and backtracking info when required.

The unvisited neighbors are explored and if none are left then the algorithm backtracks to the previous cell.

The grey color shows the backtracked cells and the pink color the solution path.



ANALYSIS

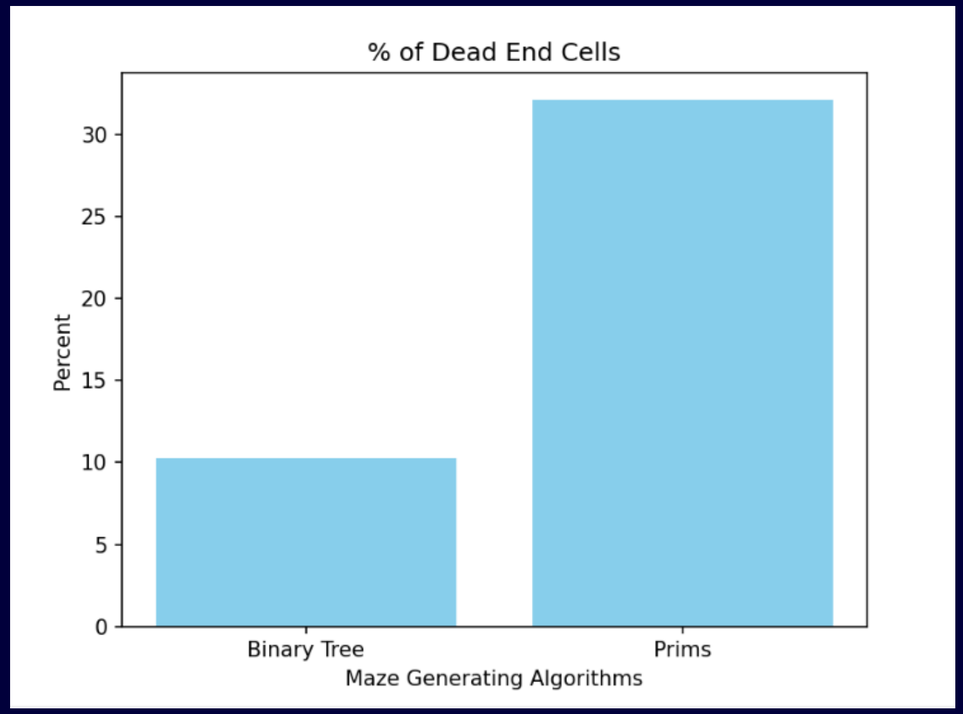
MAZE GENERATION ALGORITHMS

1. DEAD END CELLS

Method: Both algorithms were run 100 times on a 32x32 grid. The resulting maze was stored as a grid of cells. The summation of Each cell in the grid with 3 walls (and one link) gave the number of dead-end cells. This was then expressed as a percentage of the total cells in the maze.

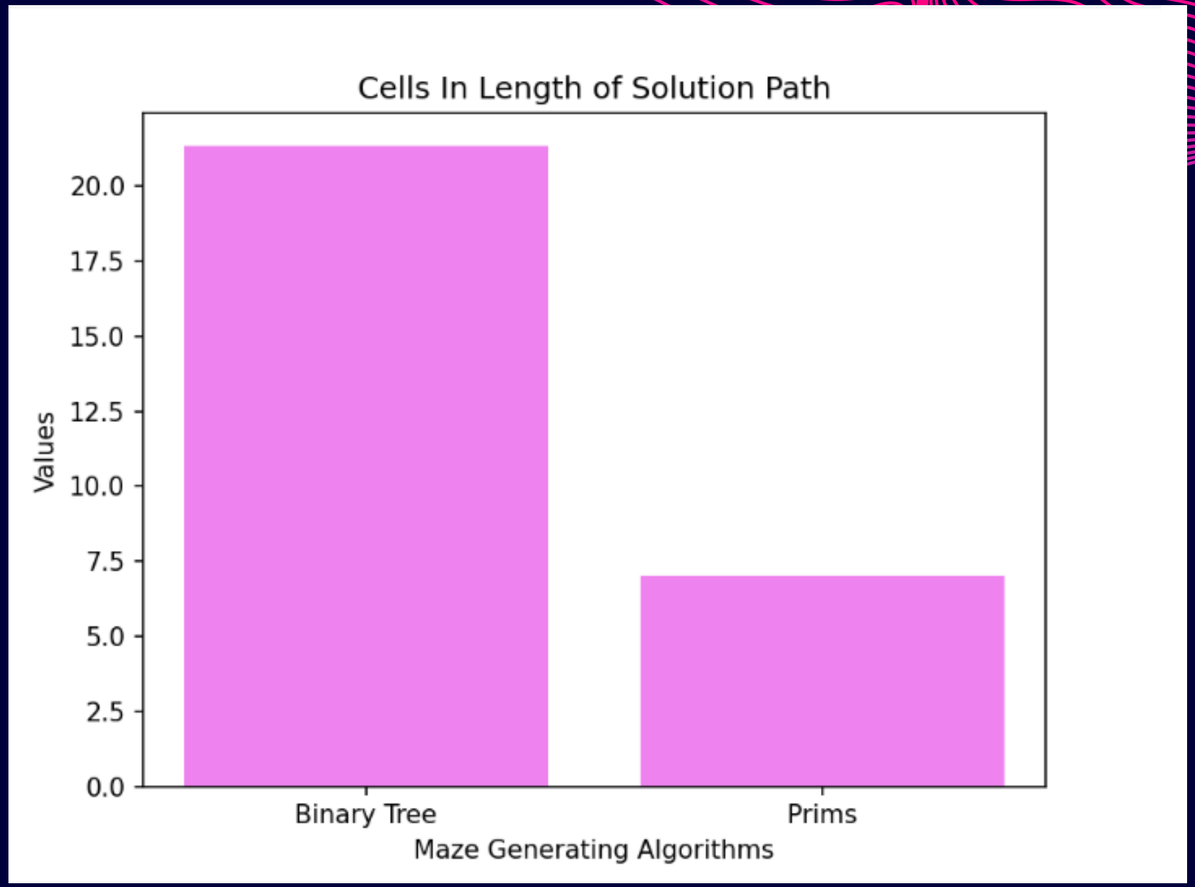
MAZE GENERATION ALGORITHMS

1. DEAD END CELLS



Prim’s Algorithm produces a significantly larger number of dead-end cells (32.12% of cells in the maze have 3 walls) than the binary tree algorithm. (10.24%)

LENGTH OF SOLUTION PATH: As per the the number of cells in the solution path is concerned, the binary tree algorithm generates a maze with significantly more cells in the solution path as compared to Prim's Algorithm, where only 7.01% of cells comprise the solution path. The former's mean percentage is around 21.33, i.e. 1/5th of cells in the maze are part of the solution path.

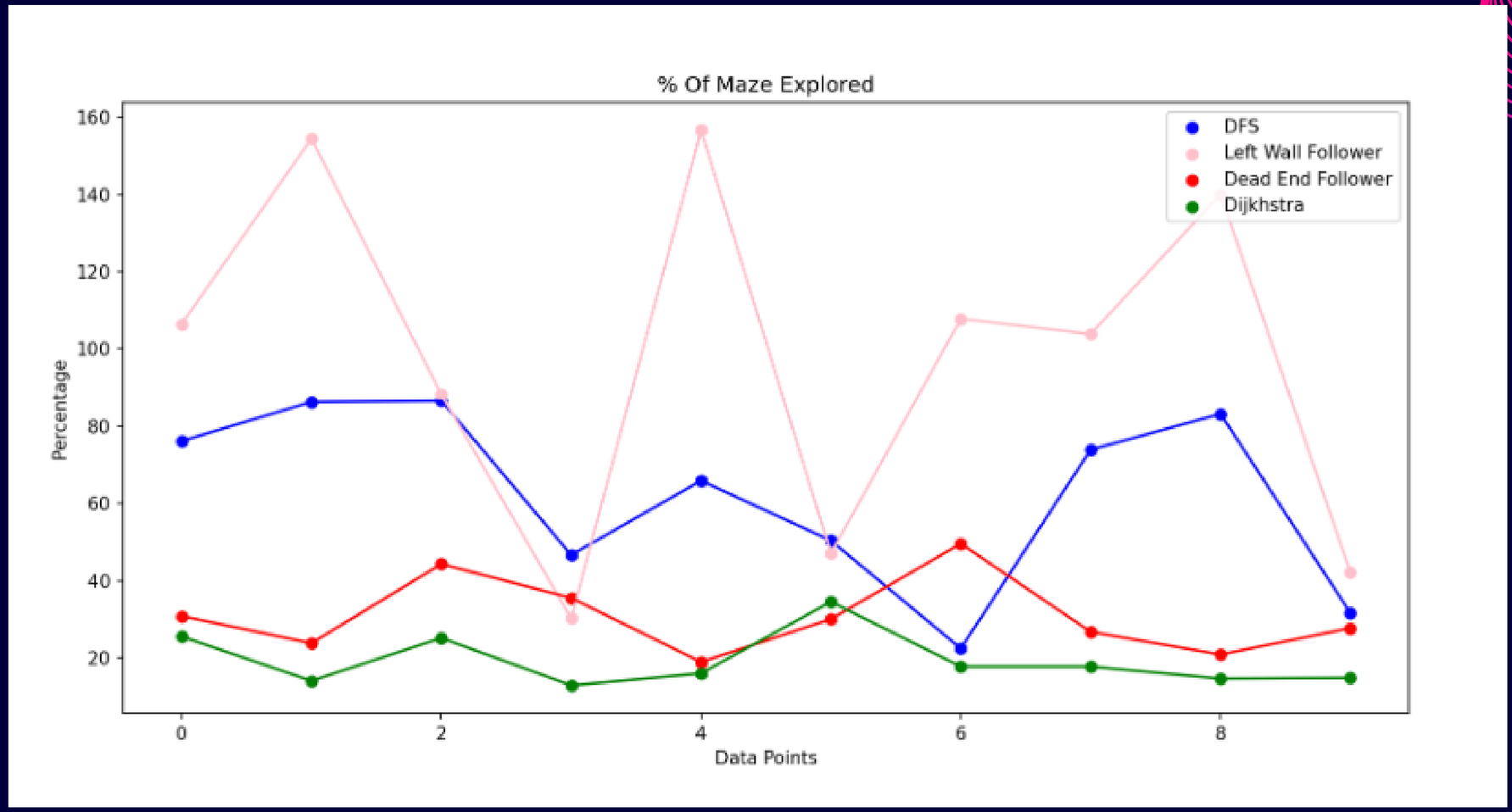


Method- DFS algorithm was used to measure the number of cells length of solution path of a 32x32 maze generated by Binary tree and Prim's algorithm. This was done 100 times, the length of solution path was expressed in terms of the percentage of maze cells in the solution path. The maze generated by both algorithms only contain single solution, therefore, any maze-solving algorithm can be used to compare the length of the path.

ANALYSIS

MAZE SOLVING ALGORITHMS

PERCENTAGE OF MAZE EXPLORED: The Left Wall Follower algorithm consistently attains a mean percentage of 99.57%, indicating its tendency to explore a substantial portion of the maze. Conversely, the Dead End and Dijkstra algorithms exhibit mean percentages of 32.70% and 19.92%, respectively, suggesting more efficient exploration with lower averages. The Dead End algorithm displays a moderately distributed exploration pattern, while Dijkstra's algorithm also showcases a relatively moderate distribution. The Depth-First Search (DFS) algorithm, with a mean of 63.99%, demonstrates a wider distribution, implying variable exploration efficiency across instances



Method: The number of cells visited while each algorithm solves a 32x32 maze was noted several times. These were expressed in terms of percentage of maze explored before the end was reached.

Github Link: https://github.com/shrutya22487/Maze_solver

THANK YOU