MAZE GAME

Artificial Intelligence Powered Maze Exploration

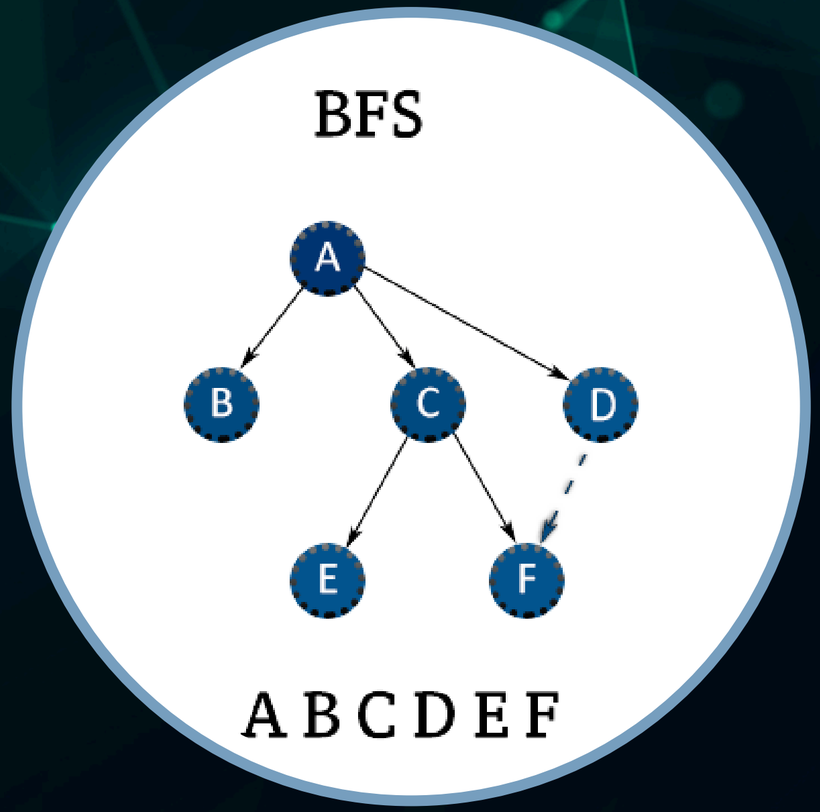
INTRODUCTION

- Maze Game generates a customizable maze and allows users to explore it using various pathfinding algorithms. Users can select an algorithm to visualize the optimal path through the maze.
- Users can implement various AI algorithms, such as Breadth-First Search, Depth-First Search, A* Search, and Greedy best first search Algorithm, to find the optimal path through the maze.
- The project provides a dynamic, real-time visualization of the AI agent's exploration and decision-making process as it navigates the maze.

CHALLENGES AND COMPLEXITIES

- **Obstacles:** Mazes can feature a variety of obstacles, such as walls, dead ends that challenge the AI agent's ability to find the optimal path.
- **Increasing Levels of Complexity:** As users select different algorithms, they'll experience how each one navigates the maze, showcasing the strengths and weaknesses of methods like **Breadth-First Search**, **Depth-First Search**, **A***, and Greedy best first search **Algorithm**.
- **Path Selection:** The maze includes strategic obstacles and pathways that test each algorithm's efficiency in finding the optimal path. Users can see firsthand how each algorithm deals with obstacles and chooses paths under different conditions.

BREADTH-FIRST SEARCH (BFS) ALGORITHM



Time Complexity: O(b^d)

1. Explore Neighbors

BFS systematically explores all the neighboring nodes at the present depth before moving on to the nodes at the next depth level.

2. Maintain Queue

The algorithm uses a queue data structure to keep track of the nodes to be visited, ensuring an organized and efficient exploration.

3. Find Shortest Path

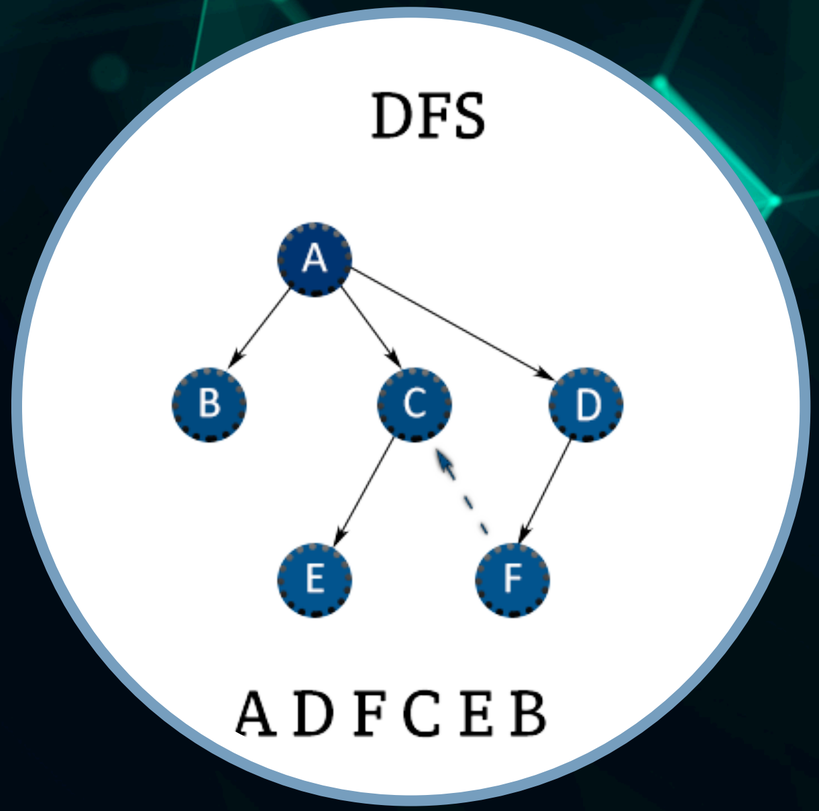
BFS is known for its ability to find the shortest path between the starting point and the goal, making it a suitable choice for maze navigation.

DEPTH-FIRST SEARCH (DFS) ALGORITHM

1. **Explore Deeper**
   DFS prioritizes exploring a single path as deeply as possible before backtracking and exploring other branches of the maze.

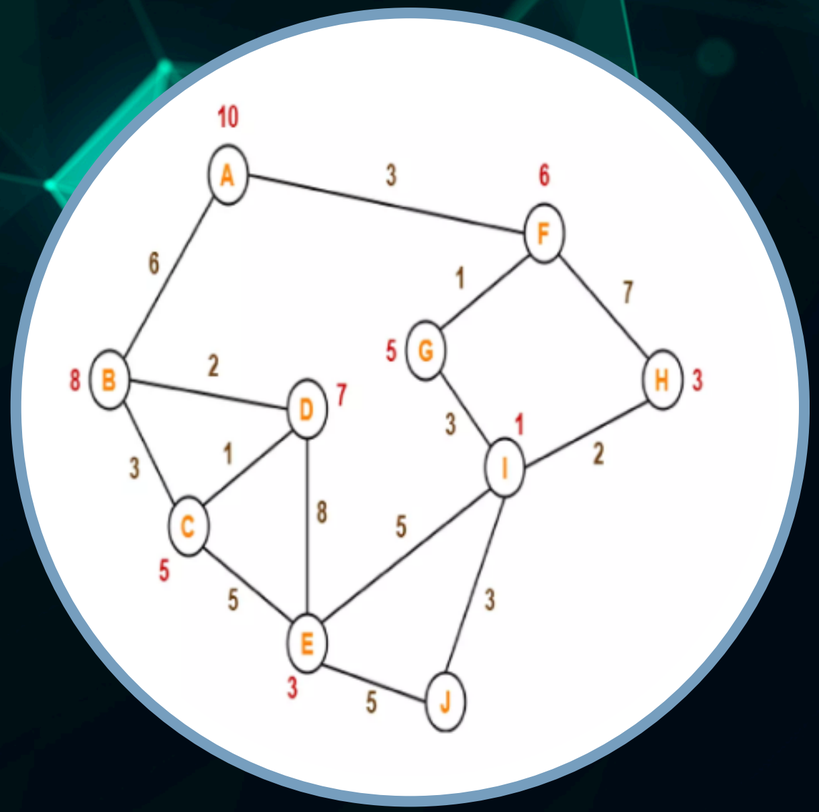
2. **Use Stack**
   The algorithm utilizes a stack data structure to keep track of the nodes to be visited, enabling efficient depth-first exploration.

3. **Find Possible Paths**
   DFS is effective in finding all possible paths through a maze, but may not necessarily identify the shortest route.



Time Complexity: O(b^m)

A* SEARCH ALGORITHM



Time Complexity: O(b^d)

1. **Heuristic Guidance**

A* Search uses a heuristic function to estimate the distance to the goal, guiding the algorithm towards the most promising path.

2. **Efficient Exploration**

The algorithm balances exploration and exploitation, efficiently navigating the maze while minimizing the number of steps required.

3. **Optimal Path**

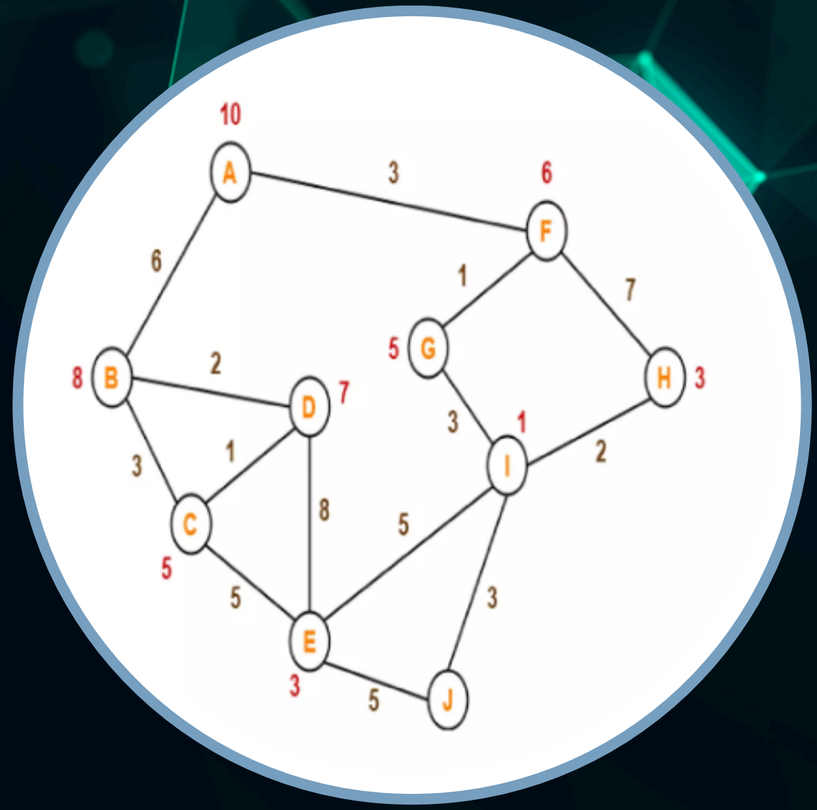
A* Search is guaranteed to find the shortest path to the goal, making it a powerful choice for maze navigation tasks.

GREEDY BEST FIRST SEARCH ALGORITHM

1. Explore Closer Paths First
Greedy Best-First Search prioritizes cells that are closest to the goal according to a heuristic, focusing on reaching the target quickly.

2. Use Priority Queue
The algorithm uses a priority queue (or heap) to manage cells, selecting the next cell to explore based on its estimated distance to the goal.

3. Efficient but Not Always Optimal
Greedy Best-First Search is efficient for finding a path quickly but may not find the shortest path, as it only considers the heuristic and not the path cost.



Time Complexity: O(b^d)

WORKING OF PYAMAZE

1. **Algorithm Selection:** The user begins by choosing the pathfinding algorithm they wish to visualize from options like Breadth-First Search (BFS), Depth-First Search (DFS), A*, or Greedy best first search Algorithm.

2. **Maze Generation:** A pre-built maze with varying complexity is generated, featuring obstacles and pathways that challenge each algorithm's pathfinding strategy.

3. **Real-Time Visualization:** The chosen algorithm is then executed, allowing the user to observe its step-by-step navigation and pathfinding process through the maze in real-time.

REAL-WORLD APPLICATIONS

ROBOTICS

The AI algorithms implemented in Pyamaze can be applied to real-world robot navigation, enabling autonomous exploration and path-finding in complex environments.

GAME DEVELOPMENT

The maze exploration and AI algorithms in Pyamaze can be leveraged to create engaging and challenging levels in video games.

URBAN PLANNING

The project's capabilities can be adapted to simulate and optimize transportation networks, emergency response, and other urban planning scenarios.

CONCLUSION

- **Versatile AI-Powered Pathfinding and Educational Tool:** Pyamaze showcases the power of AI by applying pathfinding algorithms to solve complex maze challenges and serving as a hands-on tool for learning how AI navigates and makes decisions in complex environments.
- **Interactive Visualization:** By allowing users to see real-time algorithmic exploration and decision-making, Pyamaze provides an engaging, hands-on approach to studying AI techniques in action.
- **Future Enhancements:** The project aims to expand its functionality by integrating advanced AI techniques, such as machine learning-based path optimization and adaptive algorithms for maze-solving.
- **Dynamic and Collaborative Exploration:** Future developments will include multi-agent collaboration, dynamic maze adjustments, and interactive environments to expand AI's capabilities in navigation and problem-solving.

THANK YOU!

By:
Om Rope (BT22CSA059)
Abhishek Yadav (BT22CSA007)