DATA STRUCTURE AND ALGORITHM WITH JAVA

GROUP-8
SECTION-B, 2ND YEAR, SEM IV

TABLE OF CONTENTS

OUR TEAM

MG HTUN TAUK WIN (THANATA-1968)

MG THAR LIN HTET (THANATA-1975)

MG ZAYAR HTET (THANATA-2007)

MA PHOO MON MYINT (THANATA-1981)

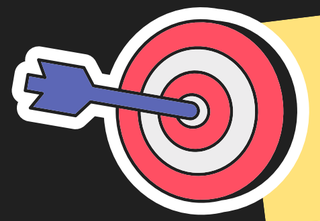
MA HSU CHAN MYAE (THANATA-1982)

INTRODUCTION

1. This project develops a maze solver using Breadth-First Search (BFS), Depth-First Search (DFS), and A* algorithms. It aims to show how each algorithm finds a path through a maze.

2. The project allows users to visualize and interact with these algorithms, helping to understand their distinct approaches to solving mazes.

3. Maze solving is a classic problem that effectively demonstrates these algorithms' practical applications and differences.



PROJECT OBJECTIVE

- Implement Maze Solving Algorithms:

  Develop solutions for mazes using three algorithms: BFS, DFS, and A*.

- Provide an Interactive Interface:

  Create a user-friendly platform where users can input mazes and choose an algorithm to solve it.

- Demonstrate Algorithm Functionality

  Show how each algorithm processes the maze and reaches the solution.

- Educational Insight

  Provide insight into the operation of each algorithm and their application in pathfinding.

TOOLS AND TECHNIQUES

PROGRAMMING LANGUAGE

- JAVA

GUIS

- SWING

IDE

- ECLIPSE

DATA STRUCTURES

- **Graph**

  Represents the maze as a network of nodes connected by edges, allowing algorithms to navigate and explore different paths.

- **Queue (for BFS)**

  Manages nodes in the order they're discovered, allowing BFS to explore paths level by level.

- **Stack (for DFS)**

  Tracks nodes with the most recent node being explored first, allowing DFS to dive deep into paths before backtracking.

- **Priority Queue (for A*)**

  Sorts nodes by their estimated distance to the goal, helping A* find the shortest path efficiently by focusing on the most promising nodes.

- **Open List (for A*)**

  Contains nodes that have been discovered but not yet fully explored, allowing A* to manage which nodes to explore next.

- **Closed List (for A*)**

  Keeps track of nodes that have already been fully explored, preventing the algorithm from revisiting them.

ALGORITHMS

- **Depth-First Search (DFS)**

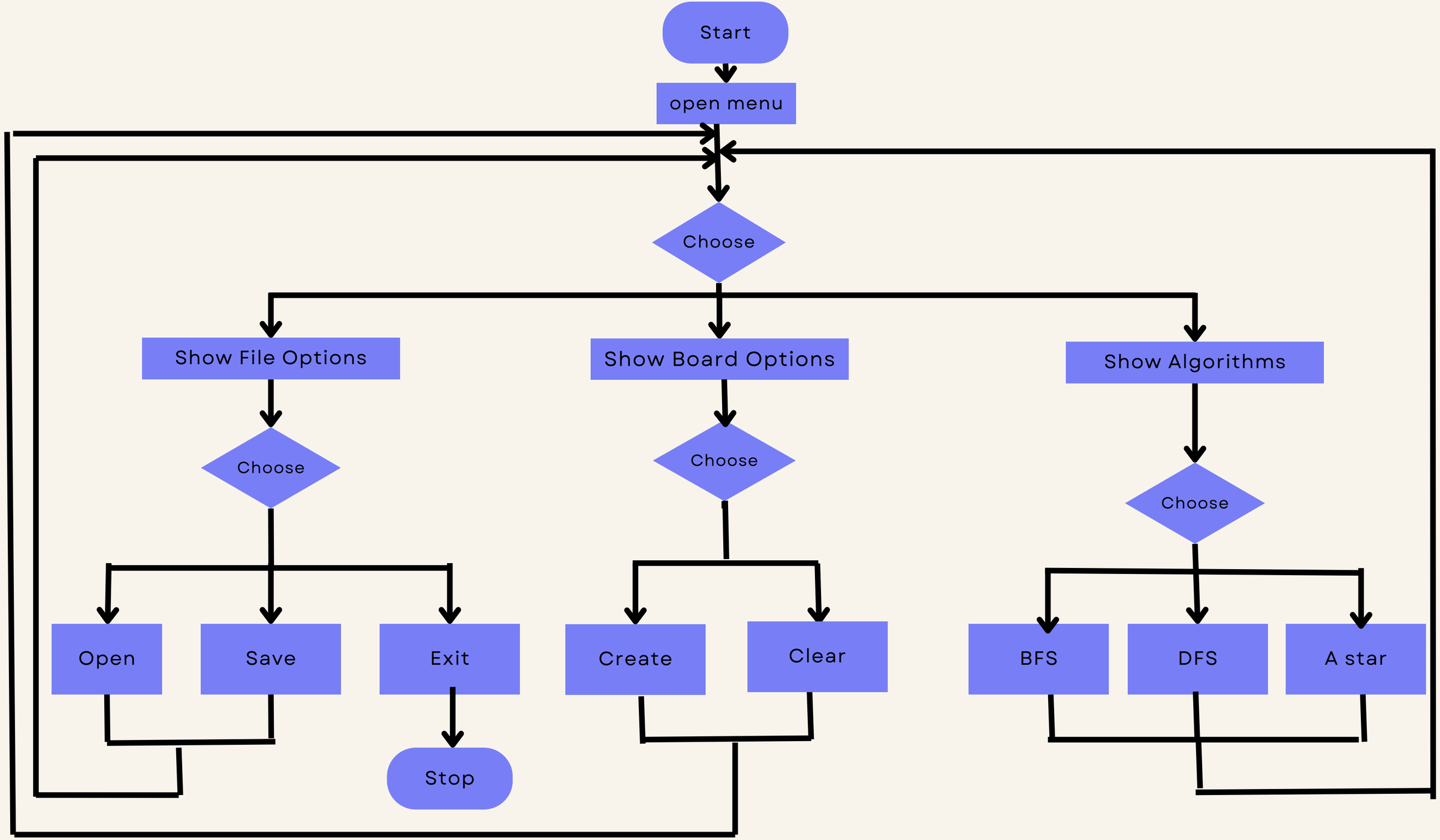
Starts from a root node and explores as far down a branch as possible before backtracking. It uses a stack to keep track of nodes to be explored. DFS systematically visits nodes, diving deep into each branch before returning to explore other branches. It is useful for exploring all possible paths in a graph or tree.

- **Breadth-First Search (BFS)**

Explores a graph level by level, starting from a root node and visiting all its neighbors before moving on. It systematically checks each node, ensuring that all are explored without revisiting. BFS is effective for finding the shortest path in unweighted graphs.

- **A* Algorithm**

Finds the shortest path by combining the cost to reach a node with an estimate of the cost to reach the goal. It uses two lists: an open list for nodes to explore and a closed list for nodes already explored. A* prioritizes nodes with the lowest total estimated cost, making it efficient for pathfinding.


Start
open menu
Choose
Show File Options
Show Board Options
Show Algorithms
Choose
Choose
Choose
Open
Save
Exit
Create
Clear
BFS
DFS
A star
Stop

TEAM MEMBERS' ROLES

GUI AND DFS

THAR LIN HTET
PHOO MON MYINT
HSU CHAN MYAE

BFS, A* AND THE REST

HTUN TAUK WIN
ZAYAR HTET

ADVANTAGES

ENHANCES PROBLEM-SOLVING SKILLS:

Develops critical thinking and problem-solving by navigating and solving complex mazes.

DEVELOP ALGORITHMIC SKILLS:

Helps users learn and apply algorithms for pathfinding and optimization.

ENCOURAGES CREATIVITY AND INNOVATION:

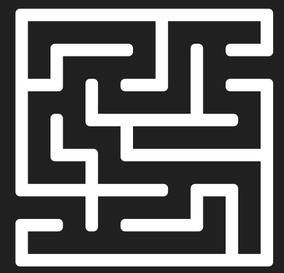
Designing varied mazes and strategies sparks creativity and explores game mechanics.

STRENGTHENS PROGRAMMING ABILITIES:

Provides practical experience with programming concepts, data structures, and logic.

OFFERS REAL-TIME FEEDBACK:

Allows users to see results in real-time, improving understanding of algorithm efficiency and effectiveness.



CONCLUSION

- This game successfully applied search algorithms such as DFS, BFS and A* algorithms to find efficient solutions to complex maze structures.
- Data structures like stacks and queues played a crucial role in managing the maze's path exploration and ensuring optimal performance.
- This project enhanced understanding of algorithmic problem-solving and provided practical experience in Java programming and debugging.
- User experience was prioritized through the development of an intuitive interface that visually demonstrates the pathfinding process.
- Overall, this project served as a valuable learning tool in both algorithm design and software development, blending theory with hands-on application.

THANK YOU FOR YOUR ATTENTION