Maze Pathfinding Final Report

Administrative

- **Team Name**: Solo Dolo
- **Team Member**: John Bagman (GitHub: @jbagman)
- **GitHub Repository**: https://github.com/JBagman11/Final-Project
- **Video Demo**: https://youtu.be/nlDNEBBnVSYo

Extended and Refined Proposal

Problem

We are trying to solve the problem of determining the most efficient and shortest path through a randomly generated maze by comparing two well-known algorithms: Breadth-First Search (BFS) and A* Search. This comparison will not only assess the correctness of the paths but also evaluate efficiency in terms of time and memory.

Motivation

Pathfinding is a core problem in many real-world applications such as robotics, navigation systems, and game AI. In each of these domains, selecting the right algorithm can lead to significant performance improvements. BFS is known for completeness but is memory-intensive, while A* is heuristic-driven and potentially faster. Comparing them helps clarify when to use which, especially under performance constraints.

Features Implemented

- Random maze generation with guaranteed solvability
- User-interactive menu with five key features
- Visualization of paths via SFML graphics
- BFS and A* execution on the same maze for comparison
- Highlighting of overlapping vs unique paths (e.g., BFS = Yellow, A* = Blue, overlap = Magenta)

Description of Data

The maze is a 2D grid of size 317x317 (100,489 cells). Each cell can be either a wall (1) or an open path (0). The start and end points are always placed on the border and are ensured to be reachable using BFS.

Tools/Languages/APIs/Libraries Used

- **Language**: C++
- **Graphics Library**: SFML (Simple and Fast Multimedia Library)
- **Standard Libraries**: vector, queue, priority_queue, tuple, chrono, cstdlib, etc.

Algorithms Implemented

- **Breadth-First Search (BFS)**: Level-order traversal using a queue. Finds the shortest path in terms of number of steps.
- *A Search (A*)**: Uses a min-heap (priority queue) and a heuristic (Manhattan distance) to prioritize exploration.

Additional Data Structures Used

- **2D Vectors**: Representing the maze grid, visited cells, parent tracking for path reconstruction.
- **Priority Queue**: Used in A* for efficient node selection.
- **Tuples and Pairs**: Representing cell coordinates and queue entries.

Distribution of Responsibilities

- **John Bagman**: Maze generation, implementation of BFS and A*, SFML visualization, performance tracking (time/memory), and report writing.

Analysis

Changes After the Initial Proposal

We initially planned to only compare the runtime and path length. However, during implementation we added:

- Memory usage estimation (based on number of steps traced)
- A fail-safe check to ensure maze solvability via BFS before allowing comparison
- A visually distinctive comparison window highlighting overlaps

This enhanced user insight and debugging.

Time Complexity Analysis

- **Maze Generation**: O(n * m), where n and m are dimensions of the maze
- **BFS Pathfinding**:
  - Worst-case time: O(V + E) → O(n * m) for grid graphs
  - Space: O(n * m) for visited and parent grids
- *A Pathfinding**:
  - Worst-case time: O((n * m) log(n * m)) due to heap operations
  - Space: O(n * m) for visited, parent, and cost tracking

The actual performance varies depending on maze density and start/end positions, but A* usually performs fewer node expansions.

Reflection

Team Experience

As a solo developer, I had full ownership of the design, implementation, debugging, and visualization. This provided deep insight into all aspects of algorithm development and performance measurement.

Challenges Faced

- Getting SFML properly configured with MinGW on Windows
- Ensuring maze solvability without manually crafting input
- Debugging A* when paths were valid in comparison but not during standalone execution (caused by shared parent structures)

Retrospective Improvements

- Would modularize the visualization layer to support headless testing
- Consider benchmarking additional algorithms (like Dijkstra)
- Improve UI for setting custom maze sizes or densities

Learning Outcomes

- Mastered BFS and A* algorithmic behavior on sparse vs dense grids
- Improved SFML integration skills for real-time visualization
- Understood how to debug pathfinding algorithms by isolating logic from rendering

---

References

1. SFML Documentation – https://www.sfml-dev.org/documentation/
2. Breadth-First Search (BFS) – GeeksforGeeks BFS Guide
3. A* Algorithm – GeeksforGeeks A* Guide
4. A* Visual Guide – RedBlobGames