JAVA MAZE GENERATOR

REQUIREMENTS ANALYSIS

Tyler Schaefer
https://github.com/TSchaefSE

Table of Contents

Abstract

This project implements a deterministic maze generation system in Java with multiple maze-solving algorithms, including Depth-First Search (DFS), Breadth-First Search (BFS), Right-Hand Search (RHS), and A* search. The system emphasizes requirements-driven design, algorithmic correctness, modular architecture, and testability. It serves as a portfolio demonstration of data structures, graph traversal strategies, and requirements-driven software engineering practices.

1. System Overview

The Maze Generator & Solver System is designed to generate perfect mazes and allow users to solve them using different algorithmic strategies. The system separates core logic from user interaction, ensuring that maze generation, traversal, and validation operate independently of presentation and user-interface concerns.

The application allows users to configure maze parameters, generate deterministic mazes, select solving algorithms, and observe solver behavior and results.

2. User Interaction Overview

The system provides a simple interaction flow that allows users to:

- Configure maze dimensions and random seed values
- Generate a new maze instance
- Select a maze-solving algorithm
- Execute the solver and view the resulting path
- Reset or regenerate the maze as needed

User interaction is intentionally minimal and focused on demonstrating algorithmic behavior rather than advanced interface features. This interaction model supports controlled experimentation with different solver strategies on identical maze instances.

3. Core System Components

Maze Generation Component

Responsible for producing perfect mazes using a deterministic algorithm. Ensures full connectivity and absence of cycles.

Maze Model

Represents the maze as a two-dimensional grid of cells. Each cell explicitly tracks wall presence in the four cardinal directions and supports traversal operations.

Solver Components

A collection of independent solver implementations that operate on the maze model:

- Depth-First Search (DFS)
- Breadth-First Search (BFS)
- Right-Hand Search (RHS)
- A* Search

Each solver can be invoked interchangeably without modifying the underlying maze representation.

4. Definitions and Terminology

Maze: A grid-based structure composed of connected cells separated by walls, where movement is restricted to defined openings between adjacent cells.

Perfect Maze: A maze in which every cell is reachable from any other cell and exactly one unique path exists between any two cells. This implies the maze is fully connected and contains no cycles.

Cell: A single unit within the maze grid that tracks wall presence in the four cardinal directions (north, south, east, west).

Path: A contiguous sequence of adjacent cells connecting a defined start location to a defined end location, traversing only through valid openings.

Solver: An algorithm that traverses the maze structure to compute a path between the start and end locations.

5. Functional Requirements

FR-1: Maze Dimension Configuration

FR-1.1: The system shall allow configuration of maze width.

FR-1.2: The system shall allow configuration of maze height.

FR-1.3: The system shall support maze dimensions as small as 1×1.

FR-2: Maze Generation Correctness

FR-2.1: The system shall generate mazes in which every cell is reachable from any other cell.

FR-2.2: The system shall generate mazes with no cycles.

FR-2.3: The system shall ensure exactly one unique path exists between any two cells.

FR-3: Deterministic Generation

FR-3.1: The system shall accept a user-defined seed for the random number generator.

FR-3.2: Given identical seed values and dimensions, the system shall generate identical maze layouts.

FR-4: Maze Representation

FR-4.1: The system shall represent the maze internally as a two-dimensional grid.

FR-4.2: Each cell shall explicitly store wall information for all four cardinal directions.

FR-4.3: The maze representation shall be accessible to solver algorithms without modification.

FR-5: Solver Support

FR-5.1: The system shall support a Depth-First Search (DFS) maze solver.

FR-5.2: The system shall support a Breadth-First Search (BFS) maze solver.

FR-5.3: The system shall support a Right-Hand Search (RHS) maze solver.

FR-5.4: The system shall support an A* maze solver.

FR-6: Solver Execution

FR-6.1: Solvers shall operate on a shared maze representation.

FR-6.2: Solvers shall not require solver-specific modifications to the maze model.

FR-6.3: Solvers shall be invokable independently of the user interface.

FR-6.4: Solvers shall return a valid, contiguous path from the start location to the end location

FR-7: Start and End Locations

FR-7.1: The system shall define a start location within the maze.

FR-7.2: The system shall define an end location within the maze.

FR-7.3: The start and end locations shall be distinct for all mazes larger than 1×1.

FR-7.4: Both the start and end locations shall be reachable from one another.

6. Non-Functional Requirements

NFR-1: Correctness

NFR-1.1: All generated mazes shall satisfy the defined perfect-maze constraints.
NFR-1.2: Solver paths shall not violate maze wall constraints.

NFR-2: Performance

NFR-2.1: Maze generation shall complete within reasonable time for maze sizes up to 100×100.
NFR-2.2: Solver execution time shall scale predictably with respect to maze size.

NFR-3: Testability

NFR-3.1: Core maze generation logic shall be independent of input/output mechanisms.
NFR-3.2: Solver logic shall be independently testable.
NFR-3.3: The system shall support automated unit testing of maze generation and solver behavior.

NFR-4: Maintainability

NFR-4.1: The system shall separate responsibilities between maze generation, maze representation, and solver logic.
NFR-4.2: Source code shall be readable and documented to support future extensions.

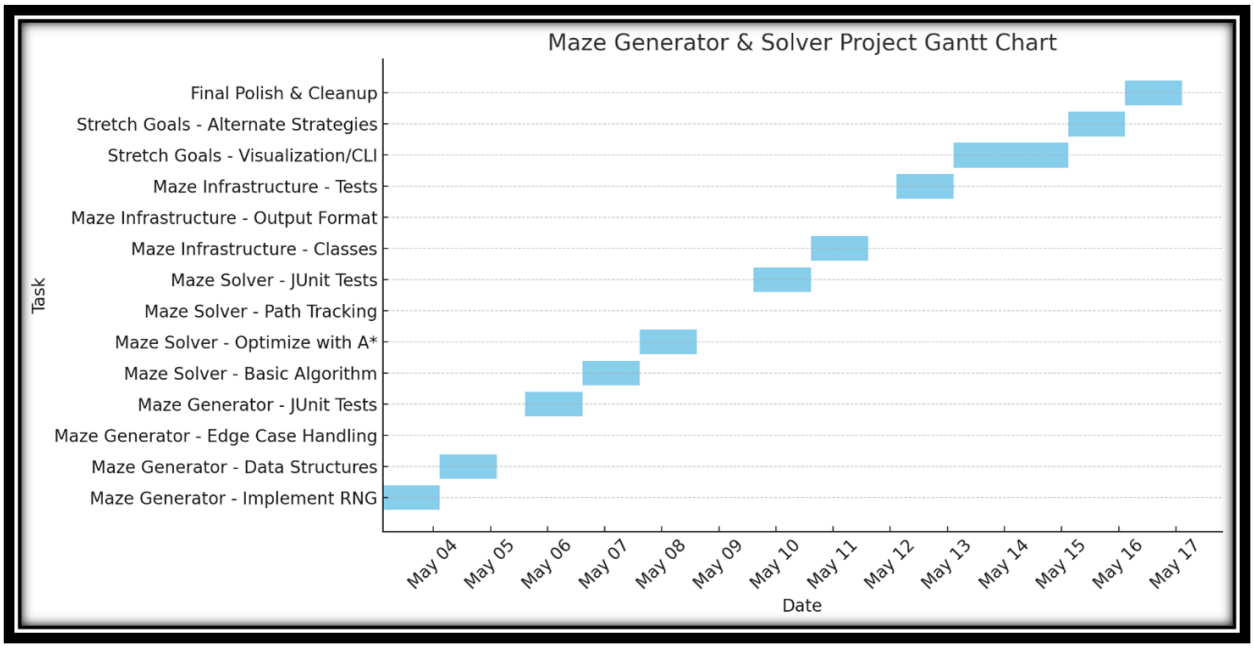
7. Validation and Testing Criteria

The system shall be considered valid when:

- All cells are reachable from the start position.
- No cycles exist in the generated maze.
- Start and end points are reachable.
- Solver paths respect maze boundaries and walls.
- Unit tests pass for edge cases, including 1×1 mazes, large mazes, and fixed seeds.

8. Project Planning and Development Timeline

Development of the Maze Generator & Solver System followed a structured, phased timeline. A Gantt chart was used to plan and track key milestones, including requirements analysis, maze generator implementation, solver development (DFS, BFS, RHS, A*), unit testing, visualization, and final polish. The chart reflects an incremental development approach emphasizing correctness, testing, and extensibility. The full timeline is provided as a supporting planning artifact in the project documentation.



9. Potential Future Enhancements

Solver performance comparison metrics:

- Additional heuristic strategies
- Exporting maze layouts to external formats
- Enhanced visualization options

10. Success Criteria

This project is successful if it demonstrates:

- Correct application of algorithmic principles
- Thoughtful system decomposition
- Deterministic and testable behavior
- Clear documentation suitable for portfolio review