# SOFTWARE REQUIREMENTS SPECIFICATION: MAZE GENERATOR

Version 1.0
October 4, 2025

## 1. INTRODUCTION

This document provides a detailed specification for the C++ Maze Generator command-line application. It outlines the project's purpose, scope, features, and constraints to guide its development and testing.

### 1.1 Purpose

The purpose of this software is to generate, display, and analyze two-dimensional mazes using different algorithms. It serves as an educational tool for learning about data structures, algorithms, object-oriented design, and build systems in C++.

### 1.2 Project Scope

The project, titled **"Recursive Maze Generation Algorithm,"** is a console-based application that will:
* Generate perfect mazes using both iterative (stack-based) and recursive depth-first search (DFS) algorithms.
* Allow users to customize maze dimensions and the random seed.
* Display the generated mazes in both Unicode and ASCII formats.
* Provide tools for comparing the performance of the two generation algorithms.
* Include a bonus feature for solving a generated maze.

The project will **not** include a graphical user interface (GUI), 3D maze generation, or network capabilities.

### 1.3 Intended Audience

This document is intended for the project developers (**Ramy, Youssef, Moaz**), testers, and the project manager. It establishes a clear agreement on the system's requirements.

### 1.4 References

- `Maze.h`
- `Maze.cpp`
- `main.cpp`
- `README.md`
- `Makefile`

## 2. OVERALL DESCRIPTION

### 2.1 Product Perspective

The Maze Generator is a self-contained, standalone command-line application. It is designed to be compiled and run on systems with a standard C++ development environment. It has no external dependencies beyond a C++17 compliant compiler and the `make` utility.

### 2.2 Product Features

The major features of the Maze Generator are:

- **Maze Generation:** Creates complex and challenging mazes.
- **Maze Customization:** Allows user control over maze size and seed.
- **Maze Visualization:** Renders the maze in the console.
- **Algorithm Comparison:** Provides tools to benchmark the iterative and recursive methods.
- **Maze Solving:** An extension feature to find a path through the maze.

### 2.3 User Classes and Characteristics

The primary users are C++ students and developers interested in learning about maze generation algorithms and software engineering principles. Users are expected to be comfortable with a command-line interface.

### 2.4 Operating Environment

The software will operate in a standard command-line terminal on operating systems that support the GNU Compiler Collection (g++) or a similar C++17-compliant compiler. The project is built using the provided `Makefile`.

### 2.5 Design and Implementation Constraints

- The application must be written in C++ and adhere to the C++17 standard [cite: 4].
- The user interface must be strictly text-based (command-line).
- The project must be compilable using the provided `Makefile` structure.
- The core logic must be encapsulated within a `Maze` class, separating it from the user interface in `main.cpp`.

## 3. SYSTEM FEATURES (FUNCTIONAL REQUIREMENTS)

This section details the specific functional requirements of the system.

### 3.1 FR-Maze-Generation

- **FR-1.1 (Iterative):** The system **shall** generate a maze using an iterative, stack-based, depth-first search algorithm.
- **FR-1.2 (Recursive):** The system **shall** generate a maze using a recursive, depth-first search algorithm.
- **FR-1.3 (Validity):** All generated mazes **shall** be "perfect" mazes, meaning there is exactly one path between any two cells, ensuring all cells are reachable.

### 3.2 FR-Maze-Customization

- **FR-2.1 (Sizing):** The user **shall** be able to specify custom integer values for the maze's width and height.
- **FR-2.2 (Seeding):** The user **shall** be able to provide an unsigned integer seed to the random number generator for creating reproducible mazes.

### 3.3 FR-Maze-Visualization

- **FR-3.1 (Unicode):** The system **shall** be able to display the maze in the console using Unicode box-drawing characters for a clean visual representation.
- **FR-3.2 (ASCII):** The system **shall** provide an option to display the maze using simple ASCII characters (# and spaces).
- **FR-3.3 (Detailed Info):** The system **shall** be able to display detailed information about the current maze, including its dimensions.

### 3.4 FR-User-Interface

- **FR-4.1 (Menu):** The system **shall** present the user with a text-based menu of numbered options to access all features.
- **FR-4.2 (Input Validation):** The system **shall** validate all numerical inputs from the user to ensure they fall within a valid, predefined range.
- **FR-4.3 (Error Handling):** The system **shall** gracefully handle invalid, non-numeric user input without crashing and prompt the user to try again.

### 3.5 FR-Algorithm-Analysis

- **FR-5.1 (Timing):** The system **shall** measure and display the time taken for maze generation in microseconds.
- **FR-5.2 (Comparison):** The system **shall** offer a function to generate two mazes of identical size and seed, one with the iterative algorithm and one with the recursive, displaying both results for comparison.
- **FR-5.3 (Benchmarking):** The system **shall** provide a performance test that benchmarks both algorithms across several predefined maze sizes (e.g., 10x10, 20x20, 30x30) and displays the results in a table.

## 4. NON-FUNCTIONAL REQUIREMENTS

This section details the non-functional requirements that constrain the system's design.

### 4.1 NFR-Performance

- **NFR-1.1 (Efficiency):** Maze generation for common sizes (up to 40x40) should feel instantaneous to the user, typically completing within 10,000 microseconds.
- **NFR-1.2 (Stack Overflow):** The recursive algorithm is expected to fail due to stack overflow on very large mazes (e.g., larger than 40x40), and this is considered acceptable behavior. The iterative algorithm should handle larger sizes without crashing.

### 4.2 NFR-Usability

**NFR-2.1 (Clarity):** The menu options and prompts must be clear and easy for a user to understand without prior instruction.

### 4.3 NFR-Maintainability

- **NFR-3.1 (Modularity):** The project is divided into a `Maze` class for core logic and a `main.cpp` for the user interface to ensure a clear separation of concerns.
- **NFR-3.2 (Code Quality):** The code must be well-commented, and functions should be properly organized to facilitate understanding and future modifications.

### 4.4 NFR-Portability

**NFR-4.1 (Compatibility):** The software must be compilable and runnable on any platform that has a C++17 compliant compiler (such as g++ or clang++) and the `make` build tool.