Software Requirements Specification

for

Trivia Maze

**Version 1.0 approved**

**Prepared by Zach Sanchez**

**Ethan Moore**

**Caleb Carroll**

**TCSS 360**

**October 16th, 2024**

**Table of Contents**

**Table of Contents ii**

**Revision History ii**

**1. Introduction 1**

1.1 Purpose 1

1.2 Document Conventions 1

1.3 Intended Audience and Reading Suggestions 1

1.4 Project Scope 1

1.5 References 1

**2. Overall Description 2**

2.1 Product Perspective 2

2.2 Product Features 2

2.3 User Classes and Characteristics 2

2.4 Operating Environment 2

2.5 Design and Implementation Constraints 2

2.6 User Documentation 2

2.7 Assumptions and Dependencies 3

**3. System Features 3**

3.1 Questions 3

3.2 Randomly Generated Mazes

3.3 Rooms (Maze room)

**4. External Interface Requirements 4**

4.1 User Interfaces 4

4.2 Hardware Interfaces 4

4.3 Software Interfaces 4

4.4 Communications Interfaces 4

**5. Other Nonfunctional Requirements 5**

5.1 Performance Requirements 5

5.2 Safety Requirements 5

5.3 Security Requirements 5

5.4 Software Quality Attributes 5

**6. Other Requirements 5**

**Appendix A: Glossary 5**

**Appendix B: Analysis Models 6**

**Appendix C: Issues List 6**

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Reason For Changes** | **Version** |
| Zach Sanchez | 10/16/24 | Initial Draft of SRS Document | 1.0 |
| Caleb Carroll | 11/10/24 | IDE and format migration | 1.1 |

# Introduction

## Purpose

The primary goal of creating an SRS for Trivia maze is to detail the specific technical decisions, and detail the implementation for our console-based game. We will also describe the system in which we developed the application, and the required specifications (if any) to run our game. Lastly, additional details such as the architecture, behavior, definitions, and more specific details/features about Trivia maze will also be included in this document.

## Intended Audience and Reading Suggestions

This document was created for anybody who wants to take a dive into the behind-the-scenes of Trivia Maze. When reading this document, it is fine and expected that readers will jump to the sections that are relevant to their curiosity or objective. Many such details will not be required to understand our program, but will be provided.

## Project Scope

Trivia Maze was originally a console-based program meant to be operated on a PC, in which players must find their way out of a randomly generated maze by answering random trivia questions correctly, but face in-game punishments for getting questions wrong, such as a door closing. After an iteration of console-based development, the team has decided to migrate the project from a console-based game to a Vite project that uses React and Tailwind CSS.

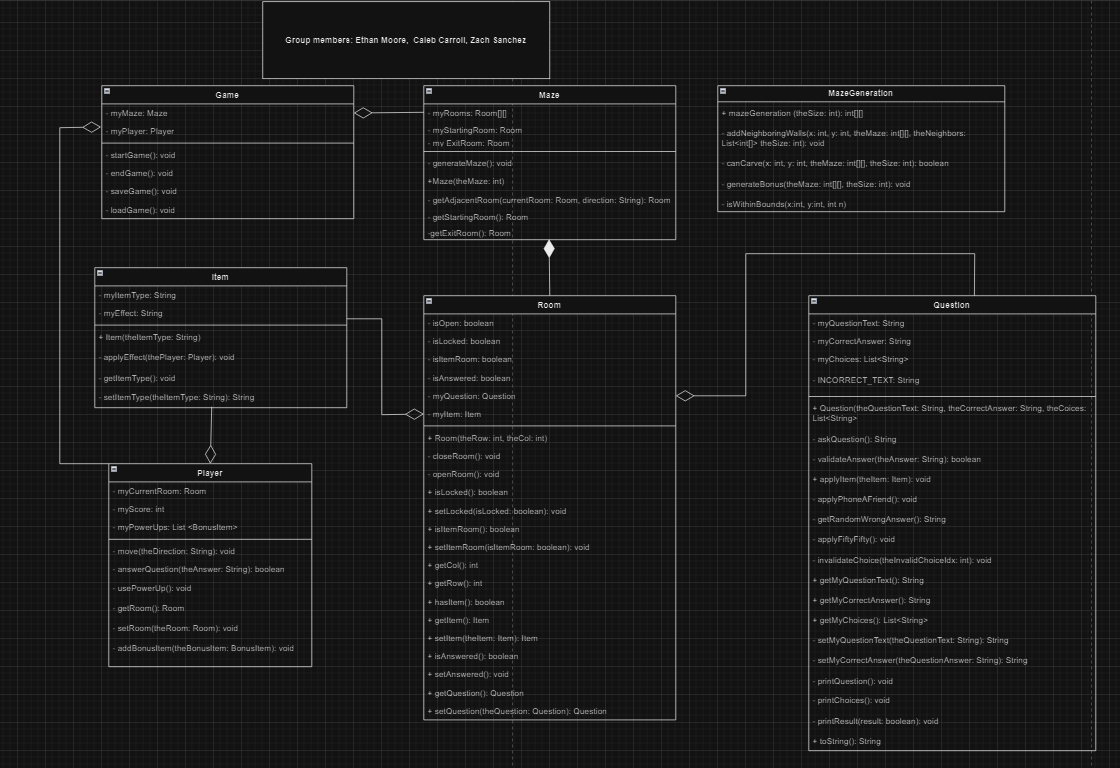
Trivia Maze was made for educational purposes from a development perspective, however is meant to be an entertaining game for users to enjoy.

# Overall Description

## Product Perspective

As briefly mentioned, Trivia maze was created for educational purposes. The project was to serve as a central application for multiple team members to contribute towards the development of in collaboration, in order to advance the skill sets of each team-member for working in a team environment. The choice to transition from java to typescript, and from console to GUI was for similar reasons.

## Product Features



Every Trivia maze playthrough features a randomly generated maze. Each randomly generated maze can be of size 4x4 to 7x7 of which the player gets to choose. Each generated maze includes random rooms, which have the chance to contain certain “bonuses”.

The interface of the maze is presented via a webpage using React, Typescript, and Tailwind CSS code. The interface also includes an interactive virtual controller which the player uses to navigate the maze.

Navigation is reliant upon answering trivia questions correctly, and the questions will be randomly chosen. If the player answers the question correctly, they will be able to move to the next room. If the player fails to do so, they will have that rooms door closed.

Playthroughs can be saved and loaded, and will maintain the relevant information for the last state of the game.

More specifics about these features will be detailed in Section 3 of this document.

## User Classes and Characteristics

This application will be primarily only for regular playthroughs of the game, which can be considered user level access. Included will be debugging and testing modes primarily meant for debugging, development, testing, and demonstrations. This access level would be considered Developer level access, and will be restricted unless enabled. Access at this level is not intended for the regular behavior or playthroughs of this game.

## Operating Environment

This application will operate on Windows PCs, and may support macs (need to double check our installer will support macs) and will be primarily tested on windows 10+ PCs and laptops. As this is a lightweight application, with smaller mazes, it should run on most modern PCs and laptops.

## Design and Implementation Constraints

The application will be written completely in Java, Typescript, and CSS Programming languages. The development team will use Visual Studio Code as the primary IDE, with version control managed via GitHub. User interfaces for this application will be developed on Visual Studio Code using Typescript and CSS. All code must adhere to the team’s established coding standards. Text based information for the questions such as the question content will come from a SQLite database.

## Assumptions and Dependencies

The operation of this application presumes users will have the required Windows 10+ PC or laptop when attempting to install and or run the application. The development/debugging flag for the game and its intended behavior, will be left disabled by default, as it is presumed the user of the application is a user with User level access, and not a developer intending to develop or debug the application. The developers presume that users will not edit, modify, or change the program in any way, and will only be using the program as intended.

# System Features

## Questions

* + 1. A question in Trivia Maze is exactly like a Question in the regular definition, except in Trivia maze, you cannot navigate to where you want to go without getting that room’s question correct. For example, if in your starting room you could go west, upon trying to navigate you would be prompted with a question. If you got the question correct, you will traverse to that open room. However, if you get the question wrong, the room closes.

## Randomly Generated Mazes

* + 1. Mazes will be randomly generated to reduce repetitive playthroughs of the game. This means details such as whether a room is open, includes an item, and even the question attached to the room could all change from game to game. For example, starting a game, you may be able to only traverse south and east, where neither open room near you contains an item, and in your next playthrough your starting room allows you to go north or west, where both rooms contain a bonus item.
  1. **Room (Maze room)**
     1. A room is an area which can contain the user, an item, and the status of open or closed. Additionally, each room has a Question, where the results of the answer to that question directly affect the navigation of users playing Trivia Maze. If a user gets a question wrong for a room, that room will change from open to closed.

# External Interface Requirements

## User Interfaces

The Trivia Maze game is operated with a graphical user interface (GUI). Users will use the left mouse button to click on a virtual controller with buttons representing cardinal directions. Using a direction key towards an unvisited room causes the player to encounter a Question which must be answered by the player via text input. If the trivia question is answered correctly, the user proceeds to the next room. An incorrectly answered question results in the door to the room the user is trying to enter being closed or potentially another type of punishment. After each input, the GUI that displays the maze will be updated reflecting said input.

## Hardware Interfaces

The game will primarily be run on Windows 10+ PCs and laptops. The hardware requirements are minimal, and any modern PC capable of running Java 8 or higher will be able to run the Trivia Maze without issue. Despite the game featuring its own custom GUI, the game should be fairly lightweight and should run on almost all modern computers.

## Software Interfaces

Trivia Maze interacts with an SQLite database for storing trivia questions. The database stores questions, possible answers, and the correct answer for each trivia event. The game retrieves these questions using SQL queries during gameplay. The game will also interact with basic Java libraries for file handling (to support the save/load feature), random generation (for creating the maze layout), and SQLite for managing the trivia question database. Future versions may extend these interactions with additional external libraries.

# Other Nonfunctional Requirements

## Performance Requirements

The performance requirements for Trivia Maze are modest, as it is uses a simple and lightweight design. The game should load fully within 5 seconds on modern PCs. Trivia questions should be fetched from the SQLite database and displayed within 1 second after the user enters a movement command. The system must be able to handle up to a four by four maze without noticeable lags or delays, even on lower-spec PCs. Response times for user commands should remain within a second to maintain fluid gameplay.

## Software Quality Attributes

Reliability is critical, and the game should not crash under normal operation conditions. It must handle invalid input gracefully by providing clear error messages and returning the user to a functional game state. The system must be easy to use, with straightforward text commands and prompts to guide the player. Ensuring that users can play the game with minimal instruction is crucial. In terms of maintainability, the code must be modular, allowing developers to easily modify or add new features, such as additional trivia questions or enhanced gameplay mechanics.

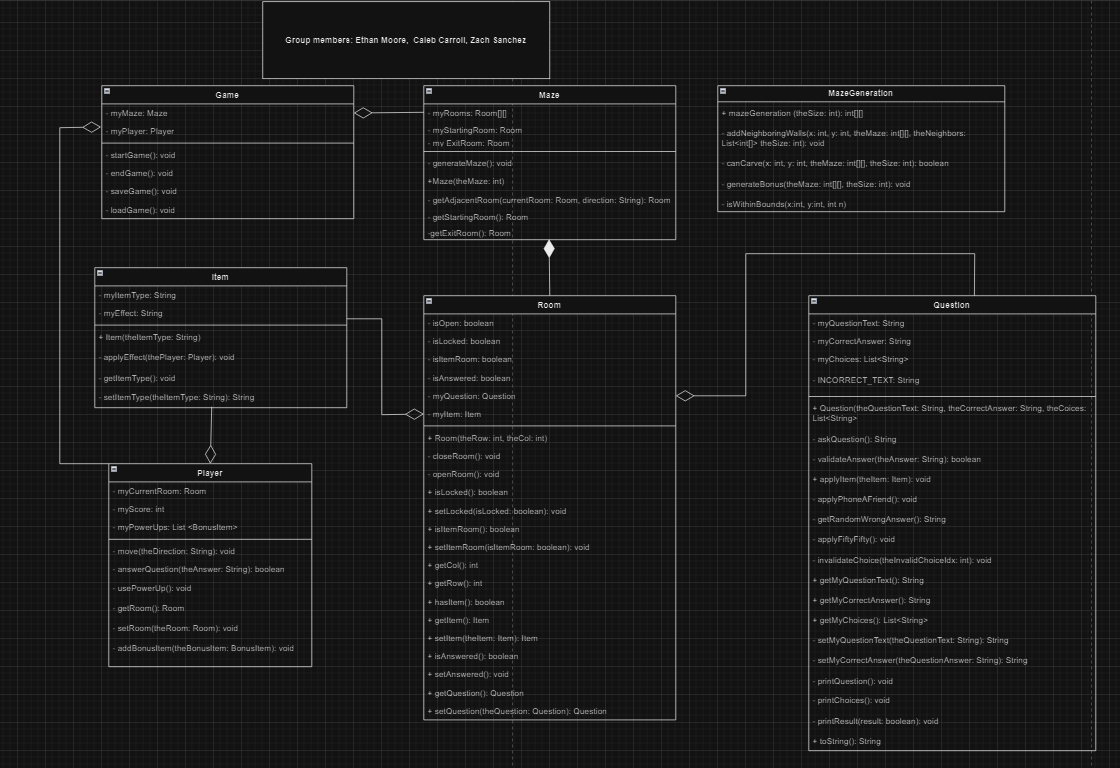
# Other Requirements

* Trivia requires facts to be verifiably true to be interesting, accurate, and fun. The source for our database will be an SQLite based database.
* Save/Load Game Functionality: The system must allow users to save their game state and load it later.
* Randomization: The trivia questions and maze generation must use reliable randomization to ensure a unique playthrough every time.

**Appendix A: Glossary**

|  |  |
| --- | --- |
| **Term** | **Definition** |
| Player | The user controlling the game, who must navigate through the maze by answering questions. |
| Room | A section of the maze where the players may encounter trivia questions. Players can only move on by answering the question correctly. |
| Maze | A randomly generated structure of interconnected rooms through which players must navigate by answering trivia questions. |
| Question Database | The SQLite database that stores the trivia questions, possible answers, and the correct answer. |
| Bonus Item | A potential item that can appear in a room, granting the player some benefit such as a hint. |

**Appendix B: Analysis Models**



**Appendix C: Issues List**

* **Issue 1: Trivia Question Database Source**

The team needs to decide whether to create a custom trivia question database or use an existing one.

* **Issue 2: Trivia Question Themes**

The team needs to decide on a theme or multiple themes for the trivia questions. Possible themes could include general knowledge, pop culture, science, history, or a specific topic tied to the maze’s design.

* **Issue 3: Design and Implementation of Power-ups**

The team is discussing the possibility of adding power-ups or penalties based on player performance. The design needs to ensure these additions enhance gameplay without unbalancing the experience.