Software Requirements Specification

for

Computer Science History Trivia Game

Version 1.1 approved

Prepared by David Chan, Nordine Chaumath, Brianna Lorraine Crowley, Boda

UW Tacoma, TCSS 360, Winter 2022 Quarter

3/14/2022

Table of Contents

Revision History

Name	**Date**	**Reason For Changes**	**Version**
Nordine Chaumath	2/21/2022	Going for basic functionality of Maze	1.1

1. Introduction

1.1 Purpose

This SRS document describes functional and nonfunctional software requirements for the 1.1 release of the Computer Science History Trivia Maze. This document will be used by members of the project team to make sure the system is functioning correctly and implements all required functionality. This Document will also be used by the customer for approval.

1.2 Document Conventions

The format of this document is a simple heading based document where the largest bold headings break off a subtopic of the document and smaller bold headings represent a subtopic of the subtopic. It is broken up into 6 main topics that each have their own subtopics. At the bottom there is a glossary, analysis model, and issue list.

1.3 Intended Audience and Reading Suggestions

Developers: This document will be used to determine what the requirements are and to make sure they are sufficiently fulfilling the requirements.

Project Manager: This document will be used as a road map for the project manager to determine what has to be done and to create an accurate timeline to completion.

Marketing Staff: This document will be used as a roadmap for the marketing team to have a detailed understanding of all the features of the Trivia Game and be able to advertise the game in the best way.

Testers: This document will be used by the tester so that at a glance they know the features that need to be tested and how to test them fully.

1.4 Project Scope

The Computer Science Trivia Maze will be a Java based application that uses text based console to generate a GUI for the ease of use from the player. This maze will be an 4x4 square where the start of the maze is at the top left and end is at the bottom right. To move from “room” to “room” the player will have to answer a randomly selected question from a SQLite database. Trivia Maze will be developed using Eclipse as the main IDE.

1.5 References

BFS - Find whether there is path between two cells in matrix - GeeksforGeeks

2. Overall Description

2.1 Product Perspective

The Computer Science History Trivia Maze is designed to create a fun way to learn and study some computer science related historical facts. This new software is expected to evolve from a simple trivia game to a full fledged stand alone interactive trivia experience.

2.2 Product Features

This program will contain an interactive trivia maze where depending on if the user got the question right they will be allowed to progress to the next room of their choice. If they got the question wrong that door will be permanently locked. It will incorporate a randomly selected question and answer pair for each room direction choice. When a question is answered incorrectly the game will lock that door. When a direction is selected it will determine what the legal directions are and once the question is answered it will check to see if the player has won the game or if there is no further path to the end meaning the player has lost.

2.3 User Classes and Characteristics

Admin: Admin class will have full permissions to add/remove questions.

Player: Players are able to only play the game.

2.4 Operating Environment

OE1 - The trivia maze game will be a stand alone application on the player's computer.

OE2 - Users will need to be on Java 8 or higher to run the game.

2.5 Design and Implementation Constraints

DIC1 - The Player must maintain an internet connection so that the maze can retrieve the questions from the database. Being that the questions are randomly selected and that there may not be enough questions in the database. There may be repeated questions.

DIC2 - A Maze of NxN is set at N = 4 in order to keep processing time low and questions database at a reasonable amount.

DIC3 - Start of maze is at top left; end of maze at bottom right.

2.6 User Documentation

Along with the software there will be a user manual that describes the game components like methods, variables, etc. and all the features. This will be delivered as an option within the game, as selecting 3 as a part of the main menu options will bring up instructions on how to play the game and the basic rules.

2.7 Assumptions and Dependencies

It is assumed that the player will have a basic knowledge of the goal and function of a maze and know the basic format of trivia questions. Another assumption is that the player will have a stable internet connection so that the maze can retrieve questions from the database. It will also be assumed that the player may have some prior experience or knowledge with computer science.

3. System Features

3.1 Maze Creation

3.1.1 Description and Priority

A player starting at the top left of a 4x4 maze room is able to “walk” to another room by answering the question correctly to move in the direction they would like to move to. The goal is to make it to the bottom right of the 4x4 maze to win. Priority = High.

3.1.2 Stimulus/Response Sequences

Stimulus: The player Chooses a room that they would like to travel through based on the available options and answers a question to pass through the room.

Response: The system queries for a random question from a question database and the player answers that question. If the player answers correctly they pass through to the desired room. If the player answers incorrectly the room is permanently locked.

3.1.3 Functional Requirements

<Omitting testing function from the Function Requirements>

TriviaMaze.Room.isLocked: Shows the state of the current room

TriviaMaze.Room.lockRoom: Makes the room unable to "walk" through after answering the trivia incorrectly.

TriviaMaze.Room.getVisit: Checks if the room has been visited for the BFS algorithm.

TriviaMaze.Room.setVisit: Sets the visitation status for BFS algorithm.

TriviaMaze.Room.setVisit: Sets the visitation status for BFS algorithm.

TriviaMaze.Room.getRow: Shows the current room's row in the 2D array

TriviaMaze.Room.getCol: Shows the current room's column in the 2D array

TriviaMaze.Maze.availableRoom: Shows the available room in order of the cardinal direction.

TriviaMaze.Maze.canMove: Checks to see if the desired direction chosen by the user is a valid direction.

TriviaMaze.Maze.move: After answering the fetched question correctly, the player will be moved to the desired room.

TriviaMaze.Maze.displayMaze: Displays the rooms in "u" = user/player, "o" = open room, "x" = locked room, and "e" = end goal to the user.

TriviaMaze.Maze.lockRoom: After failing to answer the question correctly, locks the desired room.

TriviaMaze.Maze.hasPath: A BFS algorithm to check if there is an existing path from the user to the end goal.

TriviaMaze.QuestionBean.isAsked: Checks to see if the queried question has been asked before.

TriviaMaze.QuestionBean.setAsked: Sets the status of a question**,** if the question has been seen before or not.

TriviaMaze.QuestionBean.getQuestion: Shows the question.

TriviaMaze.QuestionBean.getChoices: Shows the choices to the question asked, can be T/F or multiple choice.

TriviaMaze.QuestionBean.setChoices: Saves the user's answer to the question asked.

TriviaMaze.QuestionBean.isCorrect: Compares the database's answer to the user's answer.

TriviaMaze.QuestionBean.Display: Displays the question and available options to the fetched question

TriviaMaze.QuestionDatabaseService.iterator: iterator to move through the SQLite database.

TriviaMaze.QuestionDatabaseService.getQuestionBean: Gets a single instance of a QuestionBean class where the question has been selected at random from the database and has yet to be asked.

TriviaMaze.QuestionDatabaseService.getQuestionDataFromDatabase: Utilizes the iterator to query through each question in the database to create a hashmap of question=answer pair.

TriviaMaze.QuestionDatabaseService.gameStartUp: Creates the SQLite database by populating questions, options, and correct answer to the question in the desired table.

TriviaMaze.QuestionDatabaseService.createTable: Creates a table in the database if it does not exist yet.

TriviaMaze.QuestionDatabaseService.addQuestion: Adds additional questions, options, and answer to the question to a desired table.

3.2 System Feature 2 (and so on)

Another feature within the main menu is the usage of a cheat. By inputting 4 on the main menu, the user is taken into a menu where they can choose to activate a cheat to only solve the trivia maze in two questions rather than traversing the entire maze. This cheat can be toggled on and off through this menu, and will remain active as long as the player wants that cheat to be on when they play their next game.

4. External Interface Requirements

4.1 User Interfaces

UI-1: The Maze System will provide a help option that will describe to the user how to play the game.

UI-2: The Maze System is completely text-based.

UI-3: The Maze System has a cheats menu.

4.2 Hardware Interfaces

No hardware interfaces have been identified.

4.3 Software Interfaces

SI-1: The application communicates with a SQLite database to fetch questions to use for each door of the maze. The communication between database and application consists of only reading operations.

4.4 Communications Interfaces

The communication between the different parts is handled by the underlying operating system.

5. Other Nonfunctional Requirements

5.1 Performance Requirements

PE-1: The system shall accommodate 1 user when the program is running. It would be able to run 24 hours a day, 7 days a week, as long as the user has a connection to the internet. The estimated session duration would be between 7 and 15 minutes, depending on how big the maze is.

PE-2: The window that contains the game will appear no longer than within a possible estimated time of 15 seconds in order to load the full file and become playable to the user over a 40KBps modem connection

PE-3: Responses to questions and movement shall take no longer than 2 seconds to load after the user inputs their answer to the quiz.

PE-4: The system shall display confirmation messages to the user within 1 second after the user submits information to the system.

5.2 Safety Requirements

No safety requirements have been identified.

5.3 Security Requirements

SE-1: There should be no need to input personal information such as names, banking information, or personally identifiable information into the system.

SE-2: Users shall be required to run the program on their system and maintain a stable internet connection.

5.4 Software Quality Attributes

Availability-1: The Trivia Quiz shall be available to the user on their personal computer once they download the program, and to let the user play the game 100% of the time as long as they have an internet connection.

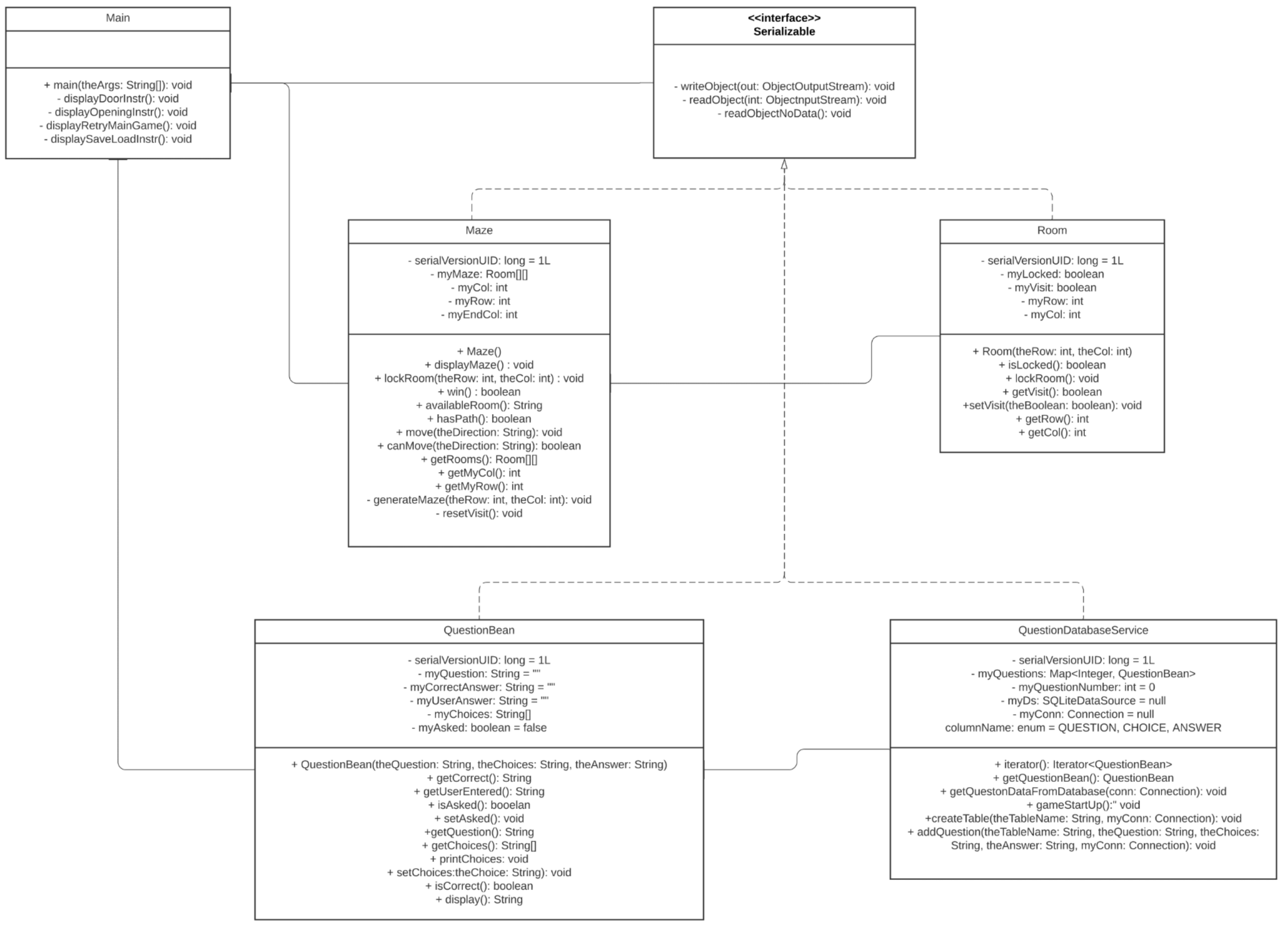
Robustness-1: If the user receives a game-over, then they'll be able to reset back to the beginning of the game.

6. Appendix

Appendix A: Glossary

Term	Definition
SRS	Software requirements and specifications.
Project manager	A member of a team that is in charge of managing the development team and how they achieve all project goals.
SQLite	A database engine that is written in C programming language.
Java	Java is a high level object oriented programming language that is similar to C and C++ in syntax.
Database	An organized electronically stored collection of data that can be searched through quickly.
BFS	Breadth First Search, a type of algorithm to check for an existing path from one point to another.

Appendix B: Analysis Models


Main
+ main(theArgs: String[]): void
- displayDoorInstr(): void
- displayOpeningInstr(): void
- displayRetryMainGame(): void
- displaySaveLoadInstr(): void
<<interface>>
Serializable
- writeObject(out: ObjectOutputStream): void
- readObject(int: ObjectnputStream): void
- readObjectNoData(): void
Maze
- serialVersionUID: long = 1L
- myMaze: Room[][]
- myCol: int
- myRow: int
- myEndCol: int
+ Maze()
+ displayMaze() : void
+ lockRoom(theRow: int, theCol: int) : void
+ win() : boolean
+ availableRoom(): String
+ hasPath(): boolean
+ move(theDirection: String): void
+ canMove(theDirection: String): boolean
+ getRooms(): Room[][]
+ getMyCol(): int
+ getMyRow(): int
- generateMaze(theRow: int, theCol: int): void
- resetVisit(): void
Room
- serialVersionUID: long = 1L
- myLocked: boolean
- myVisit: boolean
- myRow: int
- myCol: int
+ Room(theRow: int, theCol: int)
+ isLocked(): boolean
+ lockRoom(): void
+ getVisit(): boolean
+setVisit(theBoolean: boolean): void
+ getRow(): int
+ getCol(): int
QuestionBean
- serialVersionUID: long = 1L
- myQuestion: String = ""
- myCorrectAnswer: String = ""
- myUserAnswer: String = ""
- myChoices: String[]
- myAsked: boolean = false
+ QuestionBean(theQuestion: String, theChoices: String, theAnswer: String)
+ getCorrect(): String
+ getUserEntered(): String
+ isAsked(): booean
+ setAsked(): void
+getQuestion(): String
+ getChoices(): String[]
+ printChoices: void
+ setChoices:theChoice: String): void
+ isCorrect(): boolean
+ display(): String
QuestionDatabaseService
- serialVersionUID: long = 1L
- myQuestions: Map<Integer, QuestionBean>
- myQuestionNumber: int = 0
- myDs: SQLiteDataSource = null
- myConn: Connection = null
columnName: enum = QUESTION, CHOICE, ANSWER
+ iterator(): Iterator<QuestionBean>
+ getQuestionBean(): QuestionBean
+ getQuestonDataFromDatabase(conn: Connection): void
+ gameStartUp():" void
+createTable(theTableName: String, myConn: Connection): void
+ addQuestion(theTableName: String, theQuestion: String, theChoices: String, theAnswer: String, myConn: Connection): void

Appendix C: Issues List

What are the current search systems we need to use in our quiz traversal system?

Could there be a fail state that can happen in the event that the database runs out of questions?