Software Requirements Specification

For

The Eagles Trivia Maze Final Project

Version 1.0.2 approved

Prepared by Jacob Berger, Steven McConnell, Ian Oleson

Eastern Washington University CSCD350

12/8/2019

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 3
2.3 User Classes and Characteristics 3
2.4 Operating Environment 3
2.5 Design and Implementation Constraints 4
2.6 User Documentation 4
2.7 Assumptions and Dependencies 4
3. System Features 4
3.1 Player Movement 4
3.2 R etrieval Of Questions and Answers…….4
3.3 Checking of Win/Loss Conditions…….5
3.4 Save/Load Game……..……5
3.5 Add Question to Database…….……6
4. External Interface Requirements 6
4.1 User Interfaces 6
4.2 Hardware Interfaces 6
4.3 Software Interfaces 6
4.4 Communications Interfaces 6
5. Other Nonfunctional Requirements 7
5.1 Performance Requirements 7
5.2 Safety Requirements 7
5.3 Security Requirements 7
5.4 Software Quality Attributes 7
6. Other Requirements 8
Appendix A: Glossary 8
Appendix B : Use Case Diagram……..8
Appendix C : Use Case Description…….8
Appendix D: Issues List 10

Revision History

Name	Date	Reason For Changes	Version
Ian Oleson	11/3/19	Initial Draft	1.0 draft 1
Ian Oleson	12/2/19	Revisions And Final Draft	1.0.1
Ian Oleson	12/8/19	Additions and Final Draft	1.0.2

Introduction

Purpose

This SRS details and describes the software's functional and nonfunctional requirements for release 1.0 of the Team Eagles Trivia Maze Final Project. This document is intended to be used by group and those who are concerned alike to verify the correct functioning of the system. This SRS unless otherwise noted will pertain solely to release 1.0.1.

Document Conventions

As this SRS is a living document, some aspects and functionalities are subject to change throughout a lifecycle. These aspects or general systems that are not concrete will be italicized until further SRS releases higher than 1.0.1. This can include functionality and or logic ideas and will be italicized in the bolded header for that specified section.

Intended Audience and Reading Suggestions

This SRS is intended for team developers as well as the project manager, in order to map direction and functionality of different pieces of the software solution. The rest of this SRS contains an overall description of the product, i.e. the Trivia Maze (TM), will include system features such as class layouts, standards for documentation, different system requirements as well as other nonfunctional requirements. The sequence that this document should be read follows the normal convention of reading from the top and working down to the appendix and end of the document. Specifically, project managers might be more interested in the system features located in Section 3 and the overall description and design details located in Section 2.

Project Scope

The Trivia Maze is a program intended to be both educational and entertaining to the end user as they answer trivia questions, based around movie topics, to navigate through a virtual maze. The player will be given control of a character that has to interact with the environment, specifically the user will operate doors that will either open or lock depending on the validity of the answer they give based on a question. Questions can come in three forms, short answer, true or false and multiple choice. The end goal for the user is to navigate to the end of the maze, which will end the game. However, there is a loss condition that occurs when the player has locked doors that are required to finish the maze. They user will be indicated of the loss and the game will end. As this program is a game, the goal is for an engaging interface with an overall entertaining play through.

References

Wiegers, Karl. *Cafeteria Ordering System Vision and Scope Document*, www.processimpact.com/projects/COS/COS_vision_and_scope.doc
Capaul, Tom. CSCD350:Software Development, Assignment, Class Project Specifications https://canvas.ewu.edu/courses/1326315/assignments/5008777

Overall Description

Product Perspective

The Trivia Maze is a new entertainment system that does not have previous implementations for the current system. Following, this game should be a standalone product and while is intended to be of entertainment value, should be retired and decommissioned after the year of 2019. Overall, the trivia maze is a new, self-contained model that is a desktop application. An application installer will be included in the final version of the product.

Product Features

In the Trivia Maze, the user will be able to provide cardinal directions that relate to a map that is output to the screen. The user will be able to answer questions when attempting to open doors that progress them further into the maze. Also included will be the ability to enter secret cheat codes that make the game easier, such as to open all doors automatically, a God mode equivalency. Players will be given the ability to save a current games state, so if the program is exited, progress will not be completely lost, and can be loaded at a future time. Players will be given the ability to add questions to the database, so as to bolster the already existing collection of trivia questions in the database.

User Classes and Characteristics

User Classes:
All of the featured classes are favored and are important to the working of the project solution.

GameManager: Contains the main for the entire system. Sets up the games start menu as well as the start menus switches.

RunProgram: Starts the main game loop. Also takes care of loading the game's state from a previous playthrough.

PrintMaze: Handles the displaying the visual representation of the maze itself, including where the player is currently.

QuestionHandler: Implements the Serializable interface. Retrieves and displays the Question object, as well as prompts the player to answer. It then gets the player's input and determines if the question was answered correctly.

Door: Contains variables to check whether a door is locked, as well as a Question object that it is associated with. Implements the Serializable interface.

DoorTest: Contains the JUNIT tests in order to ensure valid input and functionality in regards to the Door class as it pertains to the software solution.

Maze: A four by four two-dimensional array what serves as the virtual representation of our maze. Each element in said array represents a room which the user must interact with and traverse in the goal of moving from the beginning of the maze([0][0]) to the end of the maze([3][3]) which equates

to a total of 16 possible rooms to interact with. Implements the Serializable interface.

Question: Contains a connection to a SQLite database, as well as takes in input from the user to determine whether a random questions true answer matches the input of the user. Question class also provides implementation for error checking and other issues that may arise when connecting and referencing a database. Implements the Serializable interface.

QuestionType: An enum class that specifies the types that a question may hold. These include: Multiple Choice, True False, and Short Answer.

Room: contains variables for the 4 cardinal directions, North, South, East and West, which while allow a user to choose in which direction to travel in the maze. Also includes the private member variable identifying whether a given room is an entrance or exit, verifying the status of the user and monitoring a win condition. Implements the Serializable interface.

RoomTest: Includes the JUNIT tests in order to ensure valid input and functionality in regards to this Room class as it pertains to the software solution.

Regex: This class works with the Scanner class to scrub user input and to see the validity of answers. User input will be restricted to choosing a direction in which to "move" and to answering the questions given when going through a door. Regex will be a general security measure so that our program will not accept nefarious data. Specifically, Regex ensures clean or "scrubbed" data in regards to multiple choice input, true false input, short answer input, direction input, Yes/No input and menu choice input.

RegexTest: Includes the JUNIT tests in order to ensure valid input and functionality in regards to this Regex class as it pertains to the software solution.

ScannerClass: Works in tandem with the Regex class to ensure data is well formed. All input is read as a String and then converted into the appropriate type from there. Specifically, scanner class validates the reading of Strings as input from the keyboard, the reading of characters(chars) as input from the keyboard, and the input of integers(ints) as input from the keyboard. Implements the Serializable interface.

ScannerTest: Includes the JUNIT tests in order to ensure valid input and functionality in regards to this Scanner class as it pertains to the software solution.

Player: Contains a player's name and corresponding location on the generated maze, as well as constructors, checks to see the ability of player movement, the movement and the traversal through the maze itself as well as gets and sets of pertinent information.

PlayerTest: Includes the JUNIT tests in order to ensure valid input and functionality in regards to this Player class as it pertains to the software solution.

Further Classes:
While still important, the following JUNIT test classes are non-functional to the program, and will be listed here. These all show the functionality inside the class as well as overall correctness. These JUNIT test cases are provided using a mixture of JUNIT 5, as well as JUNIT 4. Along with unitTests, included in the project are testClasses that purely show functionality of different methods in specific classes. These were for testing purposes only.

-RoomTest
-DoorTest

-Mazetest
-ScannerClassTest
-PlayerTest
-RegexTest

(Above)Fig. 1

**Fig. 1: Image of Software UML. Link referenced in appendix for further examination.*

Operating Environment

Trivia Maze is a java application that will run with all Windows Operating Systems from version Windows 7 and up. Trivia Maze will also be functional on the macOS 10.14. Otherwise, there are no major graphics or hardware limitations for this program.

Design and Implementation Constraints

The system's design, code and maintenance documentation will conform to the CSCD350TheEaglesCodeStandards [1].

User Documentation

UD-1: The System shall provide a tutorial and walkthrough that allows users to explicitly understand the win and loss conditions of the game, as well as nuisances in movement, and how to play and operate the software.
UD-2: The System shall also provide a cheat to create a "God Mode", which allows players to move through closed doors, even if the question is answered incorrectly. This cheat will be located and described in the Instruction portion of the main menu options.

Assumptions and Dependencies

AS – 1: Answers for certain questions in the trivia portion are dependent on the current date-time. This could cause answers to change over time and should be noted as time passes for this piece of software. All answers are current and up to date as of 12/3/2019.

System Features

Player Movement

Description and Priority

A Trivia Maze Player object will indicate in which direction they wish for the "character" will travel. A player may choose the direction of North, South, East or West. Players are not allowed to move through or attempt to open locked doors. Doors are locked as a result of the player answering a Trivia Question wrong (refer to 3.2). If all doors are locked for all possible rooms, player movement is terminated and the game ends. Priority = High.

3.1.2 Stimulus/Response Sequences

Stimulus: Player indicates directional movement through keyboard input.
Response: System queries maze whether this is a valid space to inhabit.
Stimulus: Player indicates an invalid directional movement through keyboard input.
Response: Player is alerted on the invalid nature of the request and is re-prompted.
Stimulus: Player indicates that they would like to add a question and answer to the database for future play through.
Response: System scans for and validates user data that is a question with a valid response. The table is then queried and the table is appended.

3.1.3 Functional Requirements

Player.moveNorth: The system shall read user input when input querying movement is presented and represent said input as movement in a the cardinal direction. Will notify if the movement is a valid input or not.
Player.moveSouth: The system shall read user input when input querying movement is presented and represent said input as movement in a the cardinal direction. Will notify if the movement is a valid input or not.

Player.moveEast: The system shall read user input when input querying movement is presented and represent said input as movement in a the cardinal direction. Will

notify if the movement is a valid input or not.

Player.moveWest: The system shall read user input when input querying movement is presented and represent said input as movement in a the cardinal direction. Will notify if the movement is a valid input or not.

Retrieval of Questions and Answers

3.2.1 Stimulus/Response

Stimulus: The player indicates that would like to answer a question in order to traverse the maze.
Response: The system collects the user data, makes sure that is well formed, and cross references the user input with the correct answer to the corresponding question. If the players input matches the answer, movement is honored. If the input is no equal to the answer, the system locks the door that corresponded to that question.

Check win/loss Conditions

3.3.1 Stimulus/Response

Stimulus: Naturally, as the player navigates through the maze, they will get some questions correctly, in turn opening doors, and they will get some questions wrong, locking some doors.
Response: An algorithm inside the maze class will access weather the maze is still solvable given the status of every door. As soon as it detects that a player is stuck and that there is no path to the end of the maze, the player will be notified.

Save/Load Game

3.4.1 Stimulus/Response

Stimulus: In the users available choices for each turn, a menu option will correspond to the ability to save to current progress in a game. At the home screen, say if a player were to quit to the main menu, a menu choice indicates whether or not the user would like to load a previous game.

Response: The system uses serialization to save the state of the current game and it is able to load on future play through.

Add Question to Database

3.5.1 Stimulus/Response

Stimulus: In the main menu of the game, the user has a choice available to add a question to the database, which can be added to the random pool that the game uses.

Response: A set of steps occur that determine what type of question they would like to ask, what they question will be and what the answer will be. This input is collected and screened, and if valid, a connection to the database is secured and is added to the database.

External Interface Requirements

User Interfaces

UI-1: The User Interface shall permit complete navigation and play through using the English Standard Keyboard alone.

UI-2: The Trivia Maze System screen displays shall always fit and conform to a fully expanded terminal or command prompt.

Hardware Interfaces

No hardware interfaces have been identified.

Software Interfaces

SI-1: Question Retrieval
SI-1.1: The Trivia Maze and Door class must interact with a functional SQlite Database in order to retrieve trivia questions and the correct corresponding answers.
SI-1.2 : Question Class must interact with the SQlite Database, in order to query the correct table with the corresponding enum class.
SI-1.3: Many core classes such as Question, ScannerClass, QuestionHandler ect. interface and use the Serializable Interface, which allows state to be saved and subsequently retrieved.

Communications Interfaces

There are no communication interfaces for this software solution. Communication interfaces might include an automatic email or subscription service, and there is no need for this in our solution.

Other Nonfunctional Requirements

Performance Requirements

PE-1: Responses to queries shall take no longer than 7 seconds to load onto the screen after the user submits the query.
PE-2: The system shall display confirmation messages to users within 4 seconds after the user submits information to the system.

Safety Requirements

No safety requirements have been identified.

Security Requirements

There are no security requirements that have been identified.

Software Quality Attributes

Robustness-1: User input for True/False and multiple choice will not have to exactly match the result in the database and will be case insensitive, this is accomplished through the use of a regular expression.

Quality of Life-1: User will be notified if the maze is no longer able to be solved. This prevents the user for playing longer than needed with no possibility of winning.

Display-1: User will be greeted by a generated computer guide, allowing a fun and interesting display for the user.

Other Requirements

International Requirements: There are no International Requirements for this piece of software at the current iteration stage.
Legal Requirements: There are no Legal Requirements for this piece of software at the current iteration stage.
Database Requirements: There are no Database Requirements for this piece of software at the current iteration stage.

Appendix A: Glossary

TM = Trivia Maze

True / False Answers = Accepted input can be case insensitive t, f, true, false.

Multiple Choice Answers = Accepted input can be case insensitive a, b, c, d.

Yes/No Answers = Accepted input can be case insensitive, such as yes,no,y,n and variations on these themes.

Appendix B: Analysis Models and Various Figures

Fig 1.

Figure 1* = This figure shows an example of a use case diagram and the ability of the user when they use the software.

Fig 2

Figure 2* shows the database schema used in the SQLite database. 3 tables are shown.

Appendix C: Use Case Description and Flow Chart

Actor: Game Player
Preconditions: Game is installed; system requirements are to the SRS documents demands.

1.) The user that will play the game, will be the player of the game.

2.) The goal of the user would be to play the game, navigating through a maze while answering questions, all while trying to complete and make it to the end of the maze.

Use Case 1	Answer Question
Actor	Player
Basic Flow	The Player is given a question out of a database that is a true-false, multiple choice or a short-answer question when they try to open a closed door.
Alternative Flow 1	The door that the Player attempts to open is not closed, no question is presented and the Player moves onto the next room.
Alternative Flow 2	If the player types in a value that is not well formed, regex classes and other methods will ensured the input is well formed and will prompt again until the input can be evaluated as correct or incorrect.
Alternate Flow 3	If a Player answers a question incorrectly, the door in front of them is locked and the question and door cannot be reattempted.

Use Case 2	Add Question To Database
Actor	Player, Database Manager
Basic Flow	In a special setting, the Player will prompt for the

	option to add a question to the database. Data is well

	formed, and the connection and table are able to be updated and the menu reappears.
Alternative Flow	In a special setting, the Player will prompt for the option to add a question to the database. The data to add to the table is not well formed and the user is re-prompted for valid input.
Alternative Flow 2	In a special setting, the Player will prompt for the option to add a question to the database. The user does not have a stable connection. This sends a error message and the menu reappears.

Use Case 3	Save Game
Actor	Player
Basic Flow	In a special setting, the Player will request that a save of state occurs. This will happen, and the Players position and game will be saved in the state it is currently in. If the Player exits, and chooses to load the from the last save, the application, will be loaded as it was left when the state was saved originally.

Use Case 4	Navigate Maze
Actor	Player
Basic Flow	Player gives a direction and goes through a doorway. They continue on this path until there is no way to complete the maze, or the maze is complete.
Alternate Flow 1	Player gives a direction that is not valid. The regex and scanner class then re-prompt until a valid direction is given.
Alternate Flow 2	Player gives a direction to a wall. The Player is then re-prompted for a direction as you cannot leave the maze by leaving the inner maze.
Alternate Flow 3	Player gives a direction to a closed, not locked, door. Player then has to answer a question to move forward (see Use Case 1).

Appendix D: Issues List

There are no apparent bugs or Issues in the current software iteration.

Appendix E: Included links

-Used for models and diagrams: https://draw.io

-Template For SRS Document :
Wiegers, Karl. *Cafeteria Ordering System Vision and Scope Document*,

www.processimpact.com/projects/COS/COS_vision_and_scope.doc