Software Requirements Specification

for

Cool Trivia Maze

Version 1.0 approved

Prepared by Renzo Aquino, Shiraz Arif, and Caleb Ernst

TCSS 360 Summer 2025

August 21st, 2025

Table of Contents

1. Introduction

This section provides an overview of the contents of this SRS document.

1.1 Purpose

This SRS document is intended for use by developers or players of the Cool Trivia Maze. This document will provide detailed information about the purpose, functional and nonfunctional requirements, and system constraints of the Cool Trivia Maze program. Additionally, relevant terms will be defined, and the internal workings of the program will be described abstractly. Unless otherwise stated, all information in this document is committed to the 1.0 release of Cool Trivia Maze.

1.2 Project Scope

Cool Trivia Maze is a simple game in which the player must answer trivia questions in order to travel through a maze, with the ultimate objective of reaching the exit of the maze. The game will not be deployed on a large scale, and is not yet available.

The purpose of the game is for learning purposes of the developers, all while providing an enjoyable product for people to experience.

The intended audience would be any person who has access to a computer and can work a computer. Since there will be references to modern internet humor, most of the target audience would be a young-adult, and potentially mature, demographic. People who want to see hentai will be given a warning xd

1.3 References

Wiegers, *Karl. Cafeteria Ordering System Vision and Scope Document*, www.processimpact.com/projects/COS/COS_vision_and_scope.doc

2. Overall Description

2.1 Product Perspective

Cool Trivia Maze is a standalone game. The game runs using Godot as a game engine, and trivia questions are stored in a database which utilizes SQLite. There is a component to connect the database (SQLite) and the game engine (Godot) together.

2.2 Product Features

In Cool Trivia Maze, players are given the objective to travel through a maze consisting of rooms in order to reach the exit located at the center of the maze. Each room contains doors leading to adjacent rooms. Doors show a different appearance to indicate whether they are *locked*, *unlocked*, or *broken*. The player may only approach doors in the same room as themself.

Doors are *locked* by default, and cannot be traveled through in this state. When approached, *locked* doors will present the player with a trivia question. If answered correctly, the door will become *unlocked*; *unlocked* doors may be freely traveled through. If answered incorrectly, the door will become *broken*; *broken* doors cannot be traveled through, and can no longer be approached. Doors which become *unlocked* or *broken* retain that state for the remainder of the playthrough and cannot be changed.

If the player succeeds in reaching the exit of the maze, they win the game. If the game reaches a state where there are no paths the player can take to reach the exit, they lose the game.

Additionally, the game features a Save/Load system, and an "Instructions" menu to provide instructions on how to play.

2.3 User Classes and Characteristics

Due to the nature of Cool Trivia Maze being a simple game, there is only one type of user that will use the program: Players. Players seek to derive entertainment from playing the game.

2.4 Operating Environment

Desktop OS: Windows 10+

Minimum HW: Dual-core CPU, 4GB RAM, 200MB disk, 1280×720 display.

Cool Trivia Maze operates using Godot 4.4.1 and MySQL.

2.5 Design and Implementation Constraints

The program must adhere to the MVC software architecture pattern. The Model layer shall encapsulate maze topology, door states, and game rules independent of rendering. The View layer shall subscribe to model change events and update visual state accordingly. The Controller layer shall translate user input (keyboard/mouse/UI widgets) into model operations (attempt move, submit answer, save, load). Observer / PropertyChange style notifications will be used between Model and View to decouple.

Additionally, the program will utilize SQLite, particularly for the purpose of handling trivia questions and answers.

2.6 User Documentation

Cool Trivia Maze will have an available "Instructions" menu to provide instructions on how to play the game. The player may choose to open or close this menu while in the main menu. User controls will be provided in the game as well.

2.7 Assumptions and Dependencies

AS-1: The user has a computer with enough storage and RAM to download and run the Cool Trivia Maze.

3. System Features

This section organizes the functional requirements for the game regarding the features of the system.

3.1 Save/Load System

3.1.1 Description and Priority

The game window of Cool Trivia Maze will contain an option panel which may be selected by the player in order to save the current game and load it at a later time.

3.1.2 Stimulus/Response Sequences

Player selects file -> save game -> game serializes the state and shows a panel confirming that the save process was successful.

3.1.3 Functional Requirements

REQ-1: The system shall persist: maze layout, player current room, per-door state.
REQ-2: Save data shall be stored in a user-writable location (default to application data directory).
REQ-3: Loading a save shall restore the persisted game state present at savetime.

3.2 Maze Navigation

3.2.1 Description and Priority

Represent the maze as a grid (9×9). Each room knows its adjacent rooms via doors (N/E/S/W). The player occupies exactly one room at a given time.

3.2.2 Stimulus/Response Sequences

The system checks the door state.

- If *locked* → present the trivia question.

- If *unlocked* → move player.
- If *broken* → deny movement (feedback message).

3.2.3 Functional Requirements

REQ-1: The system shall load a maze layout containing 81 rooms arranged in a 9×9 topology at game start (new game).

REQ-2: The system shall track one current player room at all times.

REQ-3: When the player selects a door whose state is *locked*, the system shall request a trivia question from the question database and present it to the player.

REQ-4: If the player correctly answers a trivia question from a *locked* door, the door state shall become *unlocked* and the player shall be moved through the door into the adjacent room.

REQ-5: If the player incorrectly answers a trivia question from a *locked* door, the door state shall become *broken* and the player shall remain in the current room.

REQ-6: *Broken* and *unlocked* door states shall persist for the duration of the play session, and its saved game if applicable.

REQ-7: The system shall detect a win condition when the player enters the designated exit area.

REQ-7: The system shall detect a lose condition when the player can no longer reach the designated exit area.

3.3 Trivia Question Integration

3.3.1 Description and Priority

Retrieve, randomize and present questions from a MySQL database and evaluate the players response.

3.3.2 Stimulus/Response Sequences

Maze request question for door -> QDB returns a question.

3.3.3 Functional Requirements

REQ-1: The system shall read trivia questions and answers from a local SQLite database file at startup.

REQ-2: The database will support different question formats. True and false, multiple choice, and open response answers.

REQ-3: Each question in the database will have a unique id, a type, a prompt text, and a correct answer. If the question type is not “open response”, it will also have additional answer choice(s) which are considered incorrect answers.

REQ-4: The system will have a method to select a question at random from a pool or a select type of question if the door specifies it.

REQ-5: The system will accept a player input for the corresponding question and evaluate the correctness of the answer.

REQ-6: The system will tolerate minor discrepancies (capitalization, extra whitespace) within player answers to trivia questions.

3.4 User Interface

3.4.1 Description and Priority

Retrieve, randomize and present questions from a MySQL database and evaluate the player's response.

3.42 Stimulus/Response Sequences

Maze request question for door -> QDB returns a question.

3.4.3 Functional Requirements

REQ-1: A menu system that has the following menus/choices

1. File (Save Game, Load Game, Exit)
2. Instructions/About (About, Game Play Instructions)

REQ-2: The GUI should display info about the current room.

REQ-3: An element that allows user navigation through the maze.

Only options that are valid for a room should be active/displayed.

REQ-4: A section that displays the current question. This area should be updated dynamically based on the type of question (multiple choice, T/F, short answer, etc.)

REQ-5: The *Pause* menu that must include: *Resume*, *Save Game*, *Load Game*, *Exit to Main Menu*, and three volume sliders for *Sound Effects*, *Music*, and *Master Control*.

REQ-6: When a trivia question is active, the UI shall present appropriate controls: four buttons for multiple choice, two buttons for true/false, text entry box for open response. Additionally provide a button to exit the question prompt.

4. External Interface Requirements

4.1 User Interfaces

The interface we are providing is going to be in Godot.

On startup, the user will see a title screen interface that will proceed to a save/load menu. The user can select their save file for the game at the save/load menu.

As soon as a save file is selected, the game interface will be displayed, showing the player and the maze where the user can control the player.

Each trivia question will also have an interface where the user can select their answer.

There will also be a pause screen interface where a hint screen can be accessed, showing controls and how to play the game.

4.2 Hardware Interfaces

There is no designated hardware for the game, thus it has no direct hardware interface. There will be a component that connects the database server to the system the game is being played in.

4.3 Software Interfaces

The Cool Trivia Maze game will interact with the SQLite database to get the trivia questions. Algorithms provided in Godot will interact with user interfaces for functionalities of the user interfaces. File system read and write will be used for saving. We might need an audio asset loader.

4.4 Communications Interfaces

All local; no network. Internal communication via Godot signals / observer callbacks between MVC layers. SQLite accessed via local file path.

5. Other Nonfunctional Requirements

5.1 Performance Requirements

The Cool Trivia Maze game should be able to process inputs and data quickly enough for the user to have instant and smooth control of the player (game displays).

5.2 Safety Requirements

- The Cool Trivia Maze game should make sure that antivirus programs should not detect the game as malware or a virus.
- If the game contains inappropriate material then a warning of discretion must be given.

5.3 Security Requirements

- The Cool Trivia Maze should not access any data from the user other than the save files kept in the system of the user.
- The game should also not require an internet connection to run.
- The game save files should only include the game state.

6. Other Requirements

We must document all progress and attempt to spend at least 8 story points of work each week. Each story point consists of one hour of work.

Appendix A: Glossary

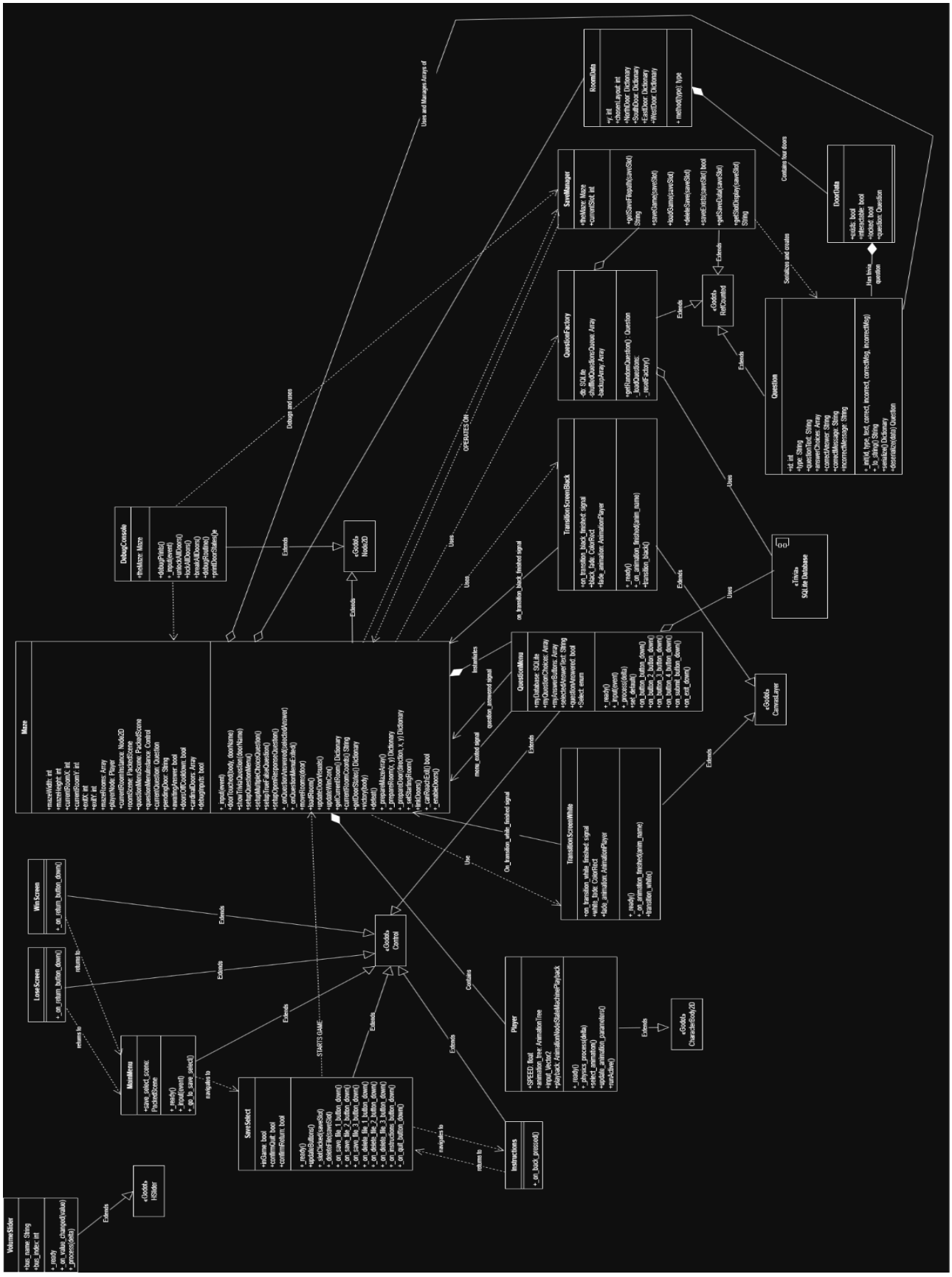
REQ-X - Requirement Number.
MVC - Model View Controller.
QDB - SQL Database.
AS-X - Assumption Number.
RAM - Random Access Memory.
GUI: Graphical user interface

Appendix B: Analysis Models

Use Case Diagram: Player interacts with Title, Game, QDB, Save/Load.

Sequence Diagram: Attempt Move -> Request Question -> Present -> Evaluate -> Update Door -> Move/Stay.

UML diagram:


VolumeSlider
+bus_name: String
+bus_index: int
+_ready
+_on_value_changed(value)
+_process(delta)
Extends
«Godot» HSlider
Maze
+mazeWidth: int
+mazeHeight: int
+currentRoomX: int
+currentRoomY: int
+exitX: int
+exitY: int
+mazeRooms: Array
+playerNode: Player
+questionMenuInstance: Node2D
+roomScene: PackedScene
+questionMenuScene: PackedScene
+questionMenuInstance: Control
+currentQuestion: Question
+pendingDoor: String
+awaitingAnswer: bool
+canPlaceExit: bool
+exitPlaced: bool
+debugmgnum: bool
+_input(event)
+doorTouched(body, doorName)
+showTriviaQuestion(doorName)
+setupQuestionMenu()
+onMultipleChoiceQuestion()
+onTrueFalseQuestion()
+onShortResponseQuestion()
+_onQuestionAnswered(isCorrect)
+_onQuestionMenuFreed()
+moveRooms(dir)
+loadRoom()
+updateDoorVisuals()
+updateMaze()
+getRoomData(x, y): Dictionary
+currentRoomControl: String
+getDoorStates(): Dictionary
+victory(body)
+debug()
+_prepareMazeArray()
+_prepareRoom(x, y): Dictionary
+_prepareRoom(x, y): Dictionary
+_setStartingRoom()
+findDoors()
+_canPlaceExit(): bool
+_setExitDoors()
DebugConsole
+theMaze: Maze
+debugPrints()
+_input(event)
+printAllDoors()
+lockAllDoors()
+breakAllDoors()
+debugUpdates()
+printDoorStateMgr
Extends
«Godot» Node2D
Debugs and uses
Uses and Manages Arrays of
RoomData
+y: int
+roomX and int
+NorthDoor: Dictionary
+SouthDoor: Dictionary
+EastDoor: Dictionary
+WestDoor: Dictionary
+ method(type): type
Contains four doors
DoorData
+public: bool
+interactable: bool
+locked: bool
+question: Question
Has a
question
SaveManager
+theMaze: Maze
+currentSlot: int
+getSaveFilepath(saveSlot) String
+saveGame(saveSlot)
+loadGame(saveSlot)
+deleteSave(saveSlot)
+saveExists(saveSlot) bool
+getSaveData(saveSlot)
+getSlotDisplay(saveSlot) String
Serializes and creates
Extends
«Godot» RefCounted
QuestionFactory
-db: SQLite
-shuffledQuestionsQueue: Array
-backupArray: Array
+getShortResponseQuestion(): Question
+_loadQuestions()
-_resetFactory()
Extends
Question
+id: int
+type: String
+questionText: String
+answerChoices: Array
+correctAnswer: String
+correctMessage: String
+incorrectMessage: String
+_init(id, type, text, correct, incorrect, correctMsg, incorrectMsg)
+to_dictionary(): String
+serialize(): Dictionary
+deserialize(data): Question
Uses
«Godot» SQLite Database
TransitionScreenBlack
+on_transition_black_finished: signal
+black_fade: ColorRect
+fade_animation: AnimationPlayer
+_ready()
+_on_animation_finished(anim_name)
+transition_black()
Extends
Uses
on_transition_black_finished signal
OPERATES On
QuestionMenu
-myDatabase: SQLite
-myAnswerChoices: Array
-myCorrectAnswer: String
-selectedAnswerText: String
-questionAnswered: bool
-Select: enum
+_ready()
+_input(event)
+_process(delta)
+set_default()
+on_button_1_button_down()
+on_button_2_button_down()
+on_button_3_button_down()
+on_submit_button_down()
+on_exit_down()
Instantiates
question_answered signal
movie_ended signal
Extends
«Godot» CanvasLayer
TransitionScreenWhite
+on_transition_white_finished: signal
+white_fade: ColorRect
+fade_animation: AnimationPlayer
+_ready()
+_on_animation_finished(anim_name)
+transition_white()
Extends
Use
On_transition_white_finished signal
Player
+SPEED: float
+animation_tree: AnimationTree
+input_Vector2
+playback: AnimationNodeStateMachinePlayback
+_ready()
+_physics_process(delta)
+select_animation()
+update_animation_parameters()
+runAction()
Extends
«Godot» CharacterBody2D
Contains
MainMenu
+save_select_scene: PackedScene
+_ready()
+_input(event)
+_go_to_save_select()
navigates to
SaveSelect
+isGame: bool
+confirmDialog: bool
+confirmReturn: bool
+_ready()
+updateButtons()
+setCheckedSaveSlot()
+deleteFileSaveSlot()
+_on_save_file_1_button_down()
+_on_save_file_2_button_down()
+_on_save_file_3_button_down()
+_on_delete_file_1_button_down()
+_on_delete_file_2_button_down()
+_on_delete_file_3_button_down()
+_on_instructions_button_down()
+_on_quit_button_down()
navigates to
returns to
Instructions
+_on_back_pressed()
LoseScreen
+_on_return_button_down()
returns to
WinScreen
+_on_return_button_down()
Extends
STARTS GAME