SE 3XA3: Software Requirements Specification
Rhythm Master

Team #16, Rhythm Masters
Almen Ng, nga18
David Yao, yaod9
Veerash Palanichamy, palanicv

March 19, 2021

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	**Version**	**Notes**
March 1, 2021	1.0	Initial Document
March 8, 2021	1.1	Added anticipated changes
March 13, 2021	1.2	Added the introduction, scope and purpose
March 17, 2021	1.2	Finished module decomposition and hierarchy

1 Introduction

Rhythm Master is modeled after the the original **video game**, **Frets on Fire**. The goal of the project is to recreate the main features of the original **game** by following the software development process midst providing proper detailed documentation. Additionally, we will be programming the **game** in a modern language, C#, improving on the outdated graphics, as well as providing meaningful test cases using Unity Test Framework.

1.1 Purpose

The purpose of this document is to outline the modular structure of the system that was found using decomposition based on the principle of information hiding and to allow new project members, maintainers, and designers to easily identify parts within the software (Parnas, 1972). This document also supports the idea of high coupling and low cohesion among the modules as it helps outline how the modules are broken up to reduce module complexity which in turn increases readability and maintainability.

1.2 Scope

The Module Guide outlines the modules which are based off the requirements specified in the **Software Requirements Specification**. In addition, the external behavior of the modules outlined in this document will be specified in the **Module Interface Specification**. Anticipated and unlikely changes are documented, which are represented by "secrets" as a result of using the principle of information hiding. Finally, a use hierarchy including all the modules specified in the document would be made to outline the use relation between modules.

1.3 Acronyms, Abbreviations, and Symbols

Table 2: **Table of Abbreviations**

Abbreviation	**Definition**
FPS	Frames per second
GUI	**Graphical User Interface**
PC	Personal computer
SRS	**Software Requirements Specification**
UTF	**Unity Test Framework**

Table 3: **Table of Definitions**

Term	**Definition**
C Sharp[C#]	The programming language used in this project.
Fret Board	A vertical musical staff upon which **notes** will be displayed.
Frets on Fire	An open source **Guitar Hero** clone.
Game/ Project/ Rhythm Master	The game that will be made by Group 16.
Game track	The game track is where the gameplay happens. It consists of music track where the user interacts to score points.
Graphical User Interface	A visual representation of the program allowing user interaction
Guitar Hero	A rhythm game where users simulate playing a guitar to a music track of their choice.
Note	An indicator for a button for the **player** to press.
Pause menu	The menu the user can open during a **game track**
Player/ User	The individual playing the **game**.
Python	The programming language used in **Frets on Fire**.
Score	A numerical value quantifying the **player's** performance in their last game.
Software Requirements Specification	A document that describes what the **system** will do and the expected performance
System	The software of the **game**
Tester	An individual testing the game either via playing the game or inspecting the code.
Unity Test Framework	Automated software testing provided by the Unity game engine
Typeform	Website that builds surveys online surveys

1.4 Overview of Document

The rest of the document is organized as follows.

- Section 2 lists the anticipated and unlikely changes of the software requirements.
- Section 3 summarizes the module decomposition that was constructed according to the likely changes.
- Section 4 specifies the connections between the software requirements and the modules.
- Section 5 gives a detailed description of the modules.
- Section 6 includes two traceability matrices. One checks the completeness of the design against the requirements provided in the SRS. The other shows the relation between anticipated changes and the modules.
- Section 7 describes the use relation between modules.

2 Anticipated and Unlikely Changes

This section lists possible changes to the system. According to the likeliness of the change, the possible changes are classified into two categories. Anticipated changes are listed in Section 2.1, and unlikely changes are listed in Section 2.2.

2.1 Anticipated Changes

Anticipated changes are the source of the information that is to be hidden inside the modules. Ideally, changing one of the anticipated changes will only require changing the one module that hides the associated decision. The approach adapted here is called design for change.

AC1: The key binds for the keys the user will use to interact in the game.

AC2: The algorithm that determines when new notes should be spawned in the game.

AC3: The visual effect when buttons are pressed

AC4: The instructions to the game

AC5: How user score data is saved locally

AC6: How scoring works during gameplay

AC7: The settings for the game

AC8: The visual effect when notes are scored

AC9: The algorithm for calculating leaderboard data

2.2 Unlikely Changes

The module design should be as general as possible. However, a general system is more complex. Sometimes this complexity is not necessary. Fixing some design decisions at the system architecture stage can simplify the software design. If these decision should later need to be changed, then many parts of the design will potentially need to be modified. Hence, it is not intended that these decisions will be changed.

UC1: Input/Output devices: The game is not intended to support alternative input devices such as gaming controllers

UC2: Architecture of the system because that is heavily dependent on Unity

UC3: There will always be a source of input data external to the software.

UC4: The graphical components of the game will always be made using Unity models.

3 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 4. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

M1: Button Handler

M2: Settings Data

M3: Save File Handler

M4: Settings Menu

M5: Leaderboard

M6: Game Manager

M7: Effects

M8: Leaderboard Calculator

M9: Instructions

M10: Note Spawner

M11: Pause Menu

M12: Main Menu

M13: Collision Detector

M14: Note Scroller

M15: Score Calculator

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	Button Handler Note Scroller Main Menu Pause Menu Leaderboard Instructions Settings Menu Effects Save File Handler
Software Decision Module	Note Spawner Collision Detector Score Calculator Leaderboard Calculator Game Manager Settings Data

Table 4: Module Hierarchy

4 Connection Between Requirements and Design

The system encompassing **Rhythm Master** is designed to satisfy all of the functional and non-functional requirements that were laid out in the **SRS**. To ensure this, the system is decomposed into modules, each with clear connections to different sets of requirements. The connections between requirements and modules are shown via traceability matrices in Table 5.

Modules are decomposed such that every requirement can be traced to at least one module. Modular decomposition helps to conceptualize these links because it breaks the system down into parts which are easier to understand. The responsibilities of a single module can generally be understood from the module name.

5 Module Decomposition

Modular decomposition is done in line with the principle of "information hiding" laid out by Parnas et al. (1984). Each module is specified with a *Secret*, a *Service*, and an *Implemented By* field. The secret is a noun describing the design decision hidden by the module. The service describes the module's function without describing how it achieves it. Finally, the *Implemented By* field describes what software is used to implement the module.

5.1 Hardware Hiding Modules

N/A

5.2 Behaviour-Hiding Module

5.2.1 Button Handler (M1)

Secrets: Actions that will be done when the gameplay buttons are pressed

Services: When the the gameplay buttons are pressed, the buttons should display a visual effect and move.

Implemented By: Rhythm Master

5.2.2 Settings Data (M2)

Secrets: Data that can be modified in the settings menu

Services: This module will save and hold the data such as volume level and key binds when the user changes the settings

Implemented By: Rhythm Master

5.2.3 Save File Handler (M3)

Secrets: Name, format and structure of the data file

Services: Reads and writes game data stored in data structures via a local file.

Implemented By: Rhythm Master

5.2.4 Settings Menu (M4)

Secrets: The method to visualize the settings menu

Services: Visualizes the settings menu with interactive GUI and text.

Implemented By: Rhythm Master

5.2.5 Leaderboard (M5)

Secrets: The method to visualize the leaderboard menu

Services: Visualizes the leaderboard with a list of player scores with interactive GUI and text.

Implemented By: Rhythm Master

5.2.6 Effects (M7)

Secrets: The user's inputs.

Services: Displays feedback to the user's input regarding timing, point combos, and note types.

Implemented By: Rhythm Master

5.2.7 Instructions (M9)

Secrets: The method to visualize the instructions menu

Services: Displays the game instructions to the user.

Implemented By: Rhythm Master

5.2.8 Pause Menu (M11)

Secrets: The interface of the game the player interacts with once the game is paused

Services: Displays an overlay representing the pause menu, provides clickable buttons that resumes the game and navigate to the main menu.

Implemented By: Rhythm Master

5.2.9 Main Menu (M12)

Secrets: The interface of the game the player interacts with once the game is launched

Services: Displays the main menu, provides clickable buttons that starts the game, and navigate to the Settings Menu and Leaderboard

Implemented By: Rhythm Master

5.2.10 Note Scroller (M14)

Secrets: Manager of displaying and scrolling notes

Services: Displays and removes the notes on the screen and advancing notes at a certain tempo.

Implemented By: Rhythm Master

5.3 Software Decision Module

5.3.1 Game Manager(M6)

Secrets: The method calls to initialize modules

Services: Overarching controller, initializes all other controllers. Initiates music upon gameplay start.

Implemented By: Rhythm Master

5.3.2 Leaderboard Calculator(M8)

Secrets: The sorting algorithm to order scores from most to least

Services:

Implemented By: Rhythm Master

5.3.3 Note Spawner (M10)

Secrets: Algorithm used for timing and placement of spawned notes

Services: Calls the note scroller to spawn a note to the **game track**

Implemented By: Rhythm Master

5.3.4 Collision Detector (M13)

Secrets: Detector of collision and overlap of two objects

Services: Provides methods and functions to determine if a collision has occured

Implemented By: Rhythm Master

5.3.5 Score Calculator (M15)

Secrets: Calculator of scores the user has earned

Services: Provides methods and functions to determine the calculation of scores

Implemented By: Rhythm Master

6 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements and between the modules and the anticipated changes.

Req.	**Modules**
FR1	M6
FR2	M6, M15
FR3	M10, M6
FR4	M1
FR5	M6, M15
FR6	M6
FR7	M6, M3
FR8	M6
FR9	M6
FR10	M9
FR11	M9
FR12	M11
FR13	M11, M12, M4
FR14	M11
FR15	M4, M2
FR16	M4
FR17	M4, M2
FR18	M4
FR19	M5
FR20	M8
FR21	M5

Table 5: Trace Between Requirements and Modules

AC	Modules
AC1	M2
AC2	M10
AC3	M1
AC4	M9
AC5	M3
AC6	M15
AC7	M4
AC8	M7
AC9	M8

Table 6: Trace Between Anticipated Changes and Modules

7 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. Parnas (1978) said of two programs A and B that A *uses* B if correct execution of B may be necessary for A to complete the task described in its specification. That is, A *uses* B if there exist situations in which the correct functioning of A depends upon the availability of a correct implementation of B. Figure 1 illustrates the use relation between the modules. It can be seen that the graph is a directed acyclic graph (DAG). Each level of the hierarchy offers a testable and usable subset of the system, and modules in the higher level of the hierarchy are essentially simpler because they use modules from the lower levels.

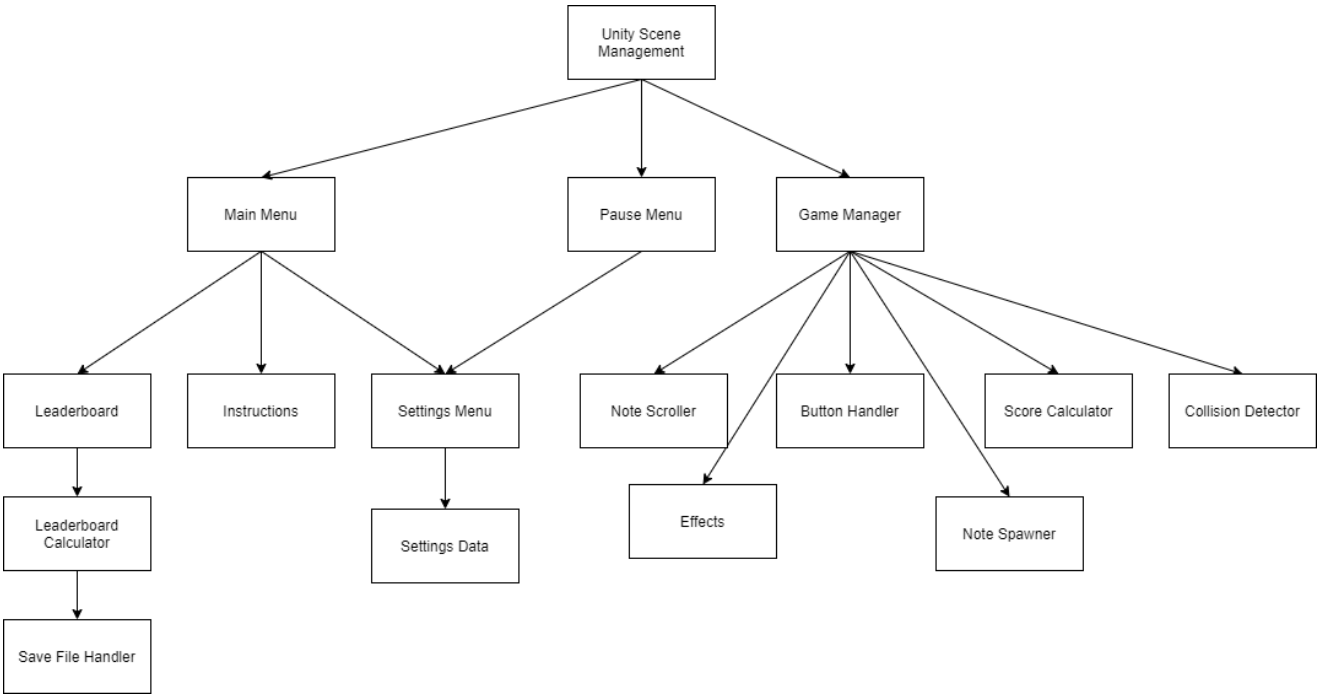

Figure 1: Use hierarchy among modules

References

David L. Parnas. On the criteria to be used in decomposing systems into modules. *Comm. ACM*, 15(2):1053–1058, December 1972.

David L. Parnas. Designing software for ease of extension and contraction. In *ICSE '78: Proceedings of the 3rd international conference on Software engineering*, pages 264–277, Piscataway, NJ, USA, 1978. IEEE Press. ISBN none.

D.L. Parnas, P.C. Clement, and D. M. Weiss. The modular structure of complex systems. In *International Conference on Software Engineering*, pages 408–419, 1984.