3XA3 Test Plan
Rhythm Master

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

March 5, 2021

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	**Version**	**Notes**
February 25, 2021	1.0	Initial Document
February 27, 2021	1.1	Finished the General Information Section
March 1, 2021	1.2	Finished the Test Plan Section
March 2, 2021	1.3	Finished Functional Requirements System Test Descriptions
March 3, 2021	1.4	Finished Non-Functional Requirements System Test Descriptions
March 4, 2021	1.5	Finished Proof of Concept, Comparison, and Unit Testing Plan
March 5, 2021	1.6	Final Draft

This document details the plan for the software testing of the **Rhythm Master** program.

1 General Information

1.1 Purpose

This test plan describes in specific terms the testing procedures that will be used to ascertain proper functioning of the **Rhythm Master** video game, as specified through its functional and non-functional requirements. The test cases within are to be used as a reference once the software is implemented and ready to use. The test structure documented here is used to minimize the possibility that a user will encounter an error during their gameplay.

1.2 Scope

The tests detailed in the plan encompass all functional and non-functional requirements set out in the **Software Requirements Specification**. In addition, the testing procedure for the proof of concept demonstration is highlighted. Finally, there is a discussion on unit tests of internal functions. The tests outlined in this document will be conducted and the document will be revised as the development of the project progresses.

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 test plan document will summarize the software, the team testing it, and the automated tools that will be used to run tests. It will then explicitly describe every test relating to an **SRS** requirement, as well as tests for the proof of concept demonstrations. Unit tests will also be described, as well as a comparison to the original implementation, **Frets on Fire**.

2 Plan

2.1 Software Description

Rhythm Master is the re-implementation of Frets on Fire, an open source Python rhythm game that simulates playing an electric guitar with gameplay strongly resembling that of **Guitar Hero**. Rhythm Master has been built using the Unity 3D engine and **C#**.

2.2 Test Team

The core test team consists of all members of Group-16 are responsible for writing and executing tests:

1. David Yao
2. Veerash Palanichamy
3. Almen Ng

In addition, once the product is in the final stages of development, volunteers would also be recruited to provide feedback on the product.

2.3 Automated Testing Approach

Automated testing will only be a minimal part of our testing plan due to the fact that the software being tested is a game which is **GUI** based. The testing of the software requires user inputs and exploration of various scenarios in the game. However, unit tests that test each game component's functionality will be automated using **Unity Test Framework**(**UTF**),

2.4 Testing Tools

One of the main unit testing tool being used is **Unity Test Framework**(**UTF**). This is a framework provided by Unity which is also the game engine **Rhythm Master**'s is developed in. **UTF** is based of NUnit library, which is an open-source unit testing library for .Net languages. **UTF** allows automated testing for both unit testing and integration testing. **Typeform** would also be used to obtain feedback from volunteer **testers**.

2.5 Testing Schedule

The testing schedule is laid out via the **Gantt chart**.

3 System Test Description

3.1 Tests for Functional Requirements

3.1.1 Gameplay Testing

1. FR-GP-1

   Type: Functional, Manual

   Initial State: Empty screen within the **Game track**

   Input: An event of starting a new **Game track**

   Output: A blank fret board

   How test will be performed: A **tester** will start a new round and check whether it starts with an empty fret board

2. FR-GP-2

   Type: Functional, Automated

   Initial State: Initialization of the **Game track**

   Input: An event of starting a new **Game track**

   Output: **Score** should be set to $INITIAL\_SCORE$

   How test will be performed: An automated test script will initialize a **Game track** and check whether the **Score** is $INITIAL\_SCORE$.

3. FR-GP-3

   Type: Functional, Manual, Dynamic

   Initial State: **Game track** should have been initialized

   Input: **Game track** has been initialized

   Output: **Notes** should be displayed

   How test will be performed: A **tester** will check whether **notes** are spawning when they start a new **Game track**.

4. FR-GP-4

   Type: Functional, Manual, Dynamic

   Initial State: **Game track** should have been initialized

   Input: **Notes** are in a state to be played

   Output: **Notes** are played

   How test will be performed: A **tester** will check whether **notes** can be played using their keyboard.

5. FR-GP-5

   Type: Functional, Manual, Dynamic

   Initial State: **Game track** should have been initialized

   Input: **Notes** are played accurately

   Output: **Tester** is awarded points

   How test will be performed: A tester can check whether their score goes up once they played a note accurately.

6. FR-GP-6

   Type: Functional, Manual

   Initial State: **Game track** should have been initialized

   Input: Initialization and changes to **score**

   Output: **score** should be displayed

   How test will be performed: A **tester** can check whether a score is being displayed on the screen

7. FR-GP-7

   Type: Functional, Manual

   Initial State: **Game track** should have been completed

   Input: **Tester** chooses the option to save their score with a username

   Output: **score** should be saved locally with the given username

   How test will be performed: A tester can complete the **Game track** and choose the option to save their score with a username. They can then check if the score is saved by viewing the leaderboard and confirming that score exists.

8. FR-GP-8

   Type: Functional, Manual, Dynamic

   Initial State: **Game track** should have been initialized

   Input: **Tester** presses an invalid key

   Output: **Tester** is not awarded points

   How test will be performed: A tester can check whether their score remains the same once they pressed an invalid key.

9. FR-GP-9

   Type: Functional, Manual

   Initial State: **Game track** should have been completed

   Input: **Tester** chooses the option to save their score with a username and enters empty username

   Output: The Game should provide a warning, and prompt their username again

   How test will be performed: A tester can complete the **Game track** and choose the option to save their score with a empty username.

3.1.2 User Interface Testing

1. FR-UI-1

   Type: Functional, Manual

   Initial State: **Game track** should have been completed

   Input: **Tester** chooses the option to redo the **Game track**

   Output: **Game track** should be reinitialized

   How test will be performed: A tester can complete the **Game track** and choose the option to redo the **Game track**. They should be able to play the **Game track** again.

2. FR-UI-2

   Type: Functional, Manual

   Initial State: **Game track** should have been completed

   Input: **Tester** chooses the option to go back to the main menu.

   Output: The game should display the main menu

   How test will be performed: A tester can complete the **Game track** and choose the option to go to the main menu. They should be shown the main menu once they choose that option.

3. FR-UI-3

   Type: Functional, Manual

   Initial State: The **tester** is viewing the main menu

   Input: Tester chooses the option to view instructions

   Output: The game should display instructions for the gameplay

   How test will be performed: A tester can choose the option to view the instruction from the main menu and check whether instructions are shown.

4. FR-UI-4

   Type: Functional, Manual

   Initial State: The **tester** is viewing the instructions

   Input: Tester chooses the option to return to the main menu

   Output: The game should display the main menu

   How test will be performed: A tester can choose the option to go to the main menu and check whether they are shown the main menu.

5. FR-UI-5

   Type: Functional, Manual, Dynamic

   Initial State: The **tester** is playing a **game track**

   Input: Tester opens the pause menu

   Output: The game should pause the game

   How test will be performed: A tester can open the pause menu while playing and check whether the game has been paused.

6. FR-UI-6

   Type: Functional, Manual, Dynamic

   Initial State: The **tester** is playing a **game track**

   Input: Tester opens the pause menu

   Output: The game should show the tester the options to opening the settings menu, going back to main menu, or restarting the **game track**.

   How test will be performed: A tester can open the pause menu while playing and check whether the options (settings menu, going back to main menu, or restarting the **game track**) are show to them.

7. FR-UI-7

   Type: Functional, Manual, Dynamic

   Initial State: The pause menu has been opened

   Input: **Tester** closes the pause menu

   Output: The game should show stop showing the pause menu and resume the gameplay.

   How test will be performed: A tester can open the pause menu while playing and close it, and check whether the game has resumed.

8. FR-UI-8

   Type: Functional, Manual, Dynamic

   Initial State: The settings menu has been opened

   Input: **Tester** specifies the volume of the game to some level using the tester interface.

   Output: The game should change the volume of the game accordingly

   How test will be performed: A tester will open the settings menu and check whether the volume of the game changes when they change it in the settings menu

9. FR-UI-9

   Type: Functional, Manual, Dynamic

   Initial State: The settings menu has been opened.

   Input: **Tester** opened the settings menu.

   Output: The game should display the version of the game.

   How test will be performed: A tester will open the settings menu and check whether the version of the game is displayed.

10. FR-UI-10

    Type: Functional, Manual, Dynamic

    Initial State: The settings menu has been opened.

    Input: **Tester** chooses to rebind their input keys to some specific key using the user interface

    Output: The game should change the input keys to the one specified by the user

    How test will be performed: A tester will open the settings menu and change the input keys. They will then play the game and check whether the new input keys are effective instead of the old one. In order to compare the implementation to that of **Frets on Fire**, every input key that is tested with **Rhythm Master** is to be tested with that game as well. Any key functioning in one game should function in the other.

11. FR-UI-11

    Type: Functional, Manual, Dynamic

    Initial State: The settings menu has been opened.

    Input: **Tester** chooses to go to the main menu

    Output: The game should display the main menu screen.

    How test will be performed: A tester will open the settings menu and choose to go back to the main menu. They can then check if they view the main menu.

12. FR-UI-12

    Type: Functional, Manual, Dynamic

    Initial State: The leaderboard screen is being displayed

    Input: **Tester** chooses to view the leaderboard

    Output: The game should display a list of players and their respective score

    How test will be performed: A tester will open the leaderboard and check whether they can view a list of players with their respective score.

13. FR-UI-15

    Type: Functional, Manual, Dynamic

    Initial State: Game is on the main menu, and there are no scores saved

    Input: **Tester** chooses to view the leaderboard

    Output: The game should display an empty list

    How test will be performed: The data structure holding high scores must be empty. To ensure that viewing an empty data structure does not raise an exception, testers must attempt to access the list of high scores while this is the case. The game should correctly display the high score table, with no values contained within.

14. FR-UI-13

    Type: Functional, Manual, Dynamic

    Initial State: The leaderboard screen is being displayed

    Input: **Tester** chooses to filter the score based on the time

    Output: The game should display a list of players and their respective score in the order that the tester chose.

    How test will be performed: A tester will open the leaderboard and check whether filtering the score by time displays the score according to that order.

15. FR-UI-14

    Type: Functional, Manual, Dynamic

    Initial State: The leaderboard screen is being displayed

    Input: **Tester** chooses to return to the main menu

    Output: The game should display the main menu screen.

    How test will be performed: A tester will open the leaderboard and check whether choosing the option to returning to main menu brings them to the main menu.

3.2 Tests for Nonfunctional Requirements

3.2.1 Look and Feel Testing

1. NFR-1-LF1

   Type: Functional, Manual, Dynamic

   Initial State: The **tester** is playing a **game track**

   Condition: Game track is unpaused and in motion

   Output: Visual effects and informational graphics produced by the game

   How test will be performed: The ability for each element shown on the screen to relay information otherwise unavailable to the tester is to be analyzed. These include information on the next note to be played, feedback on input timing, game score, and the current score multiplier. Redundant elements are to be minimized. Gameplay visuals are to be further compared to those of **Frets on Fire**, since that game is considered complete. Any additional visuals on top of those included in the original implementation are to be analyzed for necessity.

2. NFR-2-LF3

   Type: Manual, Static

   Initial State: An example image of the **fret board**

   Condition: The image is representative of a point in time during a gameplay session

   Output: Background colour in contrast with gameplay elements

   How test will be performed: **Testers** will provide their answer to question 5 of the **Usability Survey**. Their responses will be assessed to determine if the background colours are appropriate to gameplay. The background colour is also to be compared to that of **Frets on Fire**, in order to find a subjective comparison of visual clarity.

3.2.2 Usability and Humanity Testing

1. NFR-3-UH2

   Type: Functional, Manual, Dynamic

   Initial State: In game, but gameplay has not yet started

   Input: **Tester** begins gameplay

   Output: Total score after completion

   How test will be performed: Tester will be using a test **game track** of decreased difficulty, either through fewer or slower notes. They will hit every note, then compare their final score with a theoretical calculation of the maximum attainable score. The score they attain must match this value.

2. NFR-4-UH5

   Type: Functional, Manual, Dynamic

   Initial State: In game, but gameplay has not yet started

   Input: **Tester** begins gameplay

   Result: Tester has completed enough playthroughs to sufficiently understand gameplay

   How test will be performed: Testers will provide their answer to question 6 of the **Usability Survey**. Their responses will be assessed to determine if the level of difficulty is appropriate.

3.2.3 Performance Testing

1. NFR-5-PE1

   Type: Functional, Manual, Dynamic

   Initial State: In game, but gameplay has not yet started

   Input: **Tester** begins gameplay

   Output: Values of an **FPS** counter

   How test will be performed: Unity includes a built-in statistics box in its game player window. This box will be observed during gameplay to ensure framerates do not dip below $MIN\_FRAMERATE$ over the course of the **game track**. All testers must complete this test to ensure framerate is maintained across multiple computers.

2. NFR-6-PE2

   Type: Functional, Manual, Dynamic

   Initial State: In game, during gameplay

   Input: **Tester** presses any button to hit a note

   Output: Recording of gameplay

   How test will be performed: Testers will record gameplay using the screen recorder of their choice. They will make an input then view the recording frame-by-frame, observing how many frames it takes for their input to be reflected on-screen. Assuming the framerate is known, each passed frame takes a known amount of time.

3. NFR-7-PE6

   Type: Structural, Automated, Static

   Initial State: User score list has fewer than $MAX\_USER\_SCORE\_SAVES$ entries

   Input: A script using the same method to add a high score as the game itself

   Output: User score list with $MAX\_USER\_SCORE\_SAVES$ entries or more

   How test will be performed: The script will be run as many times as needed to manually fill the user score list. The test is successful if the list successfully stores $MAX\_USER\_SCORE\_SAVES$ values or more, and if those scores can all be viewed from within the game.

4. NFR-8-PE7

   Type: Structural, Manual, Dynamic

   Initial State: Game contains just the original **game track**

   Input: A new game track created by the developers

   Result: Game can be played with no difference in gameplay with the new game track

   How test will be performed: Developers will create a new game track separate from the track that already exists in the game. They will then switch the game to run on the new track. When a new game is started, the new track will be the one that is playing. The new track should be playable by the game without changing the game's behaviour in any way.

3.2.4 Operational and Environmental Testing

1. NFR-9-PE9

   Type: Functional, Manual, Dynamic

   Initial State: Game is not running.

   Condition: Computer is disconnected from the Internet.

   Result: Game runs the same as when there is an Internet connection available.

   How test will be performed: **Testers** will disconnect their system from the Internet, then launch the game and proceed with gameplay. They will attempt to access high score lists, instructions, and the **game track**. All operations should proceed identically to when the Internet connection was active.

3.3 Traceability Between Test Cases and Requirements

Table 4: **Traceability Matrix for Gameplay Requirements**

		Requirements						
		FR1	FR2	FR3	FR4	FR5	FR6	FR7
Test Cases	FR-GP-1	X						
	FR-GP-2		X					
	FR-GP-3			X				
	FR-GP-4				X			
	FR-GP-5					X		
	FR-GP-6						X	
	FR-GP-7							X
	FR-GP-8					X		
	FR-GP-9						X	

Table 5: **Traceability Matrix for UI Requirements**

		Requirements													
		FR8	FR9	FR10	FR11	FR12	FR13	FR14	FR15	FR16	FR17	FR18	FR19	FR20	FR21
Test Cases	FR-UI-1	X													
	FR-UI-2		X												
	FR-UI-3			X											
	FR-UI-4				X										
	FR-UI-5					X									
	FR-UI-6						X								
	FR-UI-7							X							
	FR-UI-8								X						
	FR-UI-9									X					
	FR-UI-10										X				
	FR-UI-11											X			
	FR-UI-12												X		
	FR-UI-13													X	
	FR-UI-14														X
	FR-UI-15												X		

Table 6: **Tracability Matrix for Non-Functional Requirements**

		Requirements								
		LF1	LF3	UH2	UH5	PE1	PE2	PE6	PE7	PE9
Test Cases	NFR-1-LF1	X								
	NFR-2-LF3		X							
	NFR-3-UH2			X						
	NFR-4-UH5				X					
	NFR-5-PE1					X				
	NFR-6-PE2						X			
	NFR-7-PE6							X		
	NFR-8-PE7								X	
	NFR-9-PE9									X

4 Tests for Proof of Concept

Our proof of concept implemented most of the gameplay features. The following are test cases related to gameplay that were conducted.

4.1 Gameplay Testing

1. POC-GP-1

   Type: Functional, Manual

   Initial State: Initialization of the **Game track**

   Input: Play button has been clicked

   Output: A blank fret board is displayed

   How test will be performed: The **tester** will start a new round and check whether it starts with an empty fret board

2. POC-GP-2

   Type: Functional, Automated

   Initial State: Initialization of the **Game track**

   Input: An event of starting a new **Game track**

   Output: **Score** should be set to $INITIAL\_SCORE$

   How test will be performed: An automated test script will initialize a **Game track** and check whether the **Score** is $INITIAL\_SCORE$.

3. POC-GP-3

   Type: Functional, Manual, Dynamic

   Initial State: **Game track** should have been initialized

   Input: **Game track** has been initialized

   Output: **Notes** should be displayed

   How test will be performed: A user will check whether **notes** are spawning when they start a new **Game track**.

4. POC-GP-4

   Type: Functional, Manual, Dynamic

   Initial State: **Game track** should have been initialized

   Input: **Notes** are in a state to be played

   Output: **Notes** are played

How test will be performed: A **tester** will check whether **notes** can be played using their keyboard.

5. POC-GP-5

   Type: Functional, Manual, Dynamic

   Initial State: **Game track** should have been initialized

   Input: **Notes** are played accurately

   Output: **Tester** is awarded points

   How test will be performed: A tester can check whether their score goes up once they played a note accurately.

6. POC-GP-6

   Type: Functional, Manual

   Initial State: **Game track** should have been initialized

   Input: Initialization and changes to **score**

   Output: **score** should be displayed

   How test will be performed: A user can check whether a score is being displayed on the screen

7. FR-GP-7

   Type: Functional, Manual, Dynamic

   Initial State: **Game track** should have been initialized

   Input: **Tester** inputs the wrong key

   Output: **Tester** is not awarded points

   How test will be performed: A tester can check whether their score remains the same once they played a note incorrectly.

5 Comparison to Existing Implementation

Rhythm Master, as of the time of the creation of this document, has implemented a skeleton mirroring the gameplay behaviour of **Frets on Fire**. Scrolling **notes** on a background **fret board** in conjunction with playing background music have all been implemented. In addition, score-keeping functionality is fully implemented, with score and score multiplier incrementation based on the amount of notes hit.

Things that remain to be implemented include a main menu, locally stored high scores, complete **game tracks**, and tangible graphical improvements over **Frets on Fire**.

6 Unit Testing Plan

6.1 Unit testing of internal functions

Unit tests of internal functions of the software are done using ASSERT statements within test scripts using the **Unity Test Framework**, each script corresponds to a module. The ASSERT statements take in certain predetermined values/statements and compare the method output against the intended output. The inputs would consist of normal, boundary, and inputs that should generate exceptions/errors. If the ASSERT statements evaluate to TRUE, it would prove the correctness of the code.

None of these test cases need stubs or drivers.

Coverage matrices will be used to track all unit tests to ensure that all requirements that can be tested with unit tests has been covered.

An important detail to note is that not all internal functions are able to be tested using unit testing and would instead be tested manually.

By the end of unit testing, $UNIT\_TEST\_COVERAGE$ percent of the code should be tested and the rest would be manual testing.

6.2 Unit testing of output files

There will be not unit testing of output files conducted as **Rhythm Master** does not output any files.

7 Appendix

7.1 Symbolic Parameters

The definition of the test cases will call for SYMBOLIC_CONSTANTS. Their values are defined in this section for easy maintenance.

$INITIAL\_SCORE = 0$
$MIN\_FRAMERATE = 30$
$MAX\_USER\_SCORE\_SAVES = 100$
$UNIT\_TEST\_COVERAGE = 30$

7.2 Usability Survey Questions

Tester feedback is an important component of testing and their experience playing **Rhythm Master** will be taken into account.

1. Subjectively, how would you compare your experience playing this game to other games, such as **Guitar Hero**?
2. On a scale from 1-5, how comfortable is the placement of the controls?
3. On a scale from 1-5, how responsive did the game feel to your inputs?
4. On a scale from 1-5, how well does the **game track** match the background music?
5. Would you consider the background colour distracting from your gameplay?
6. How many times did you need to play the game to understand its mechanics well?