**Software Requirements and Design Document**

**For**

**Group <7>**

Version 2.0

**Authors**:

Jake Nilsson

Patrick Canady

Estefano Cuna

Suyeol Ji

Brian Morales

# **Overview (5 points)**

* We are proposing a Fruit Ninja/Osu type 2D rhythm game where targets are thrown in rhythm to a

song. Targets need to be destroyed/sliced by dragging the mouse through them with proper timing.

The game will be developed in the Unity engine and will have physics-based interaction.

# **Functional Requirements (10 points)**

1. The user must be able to change the music volume with the volume sliders in each settings menu. **(Medium)**
2. The volume sliders must sync between menu and scene changes. **(Medium)**
3. The user must be able to slice through a note by holding click and dragging across the note. Additionally, the user must not be able to slice through notes when not holding a click. **(High)**
4. Notes must be ‘tossed’ from the top or bottom of the screen into the user’s view, so that the user may slice them. **(High)**
   1. Slicing notes can only generate points for the user if it’s cut when the note moves within the correct color-coded segment (e.g. the yellow note enters the yellow judgment bar) **(Medium)**
   2. There will be 4 judgment bars (red, yellow, green, and blue), and the equivalent corresponding notes. **(Medium)**
5. The user must be able to pause the game at any time by clicking a pause button on the top left, which would also freeze the current game-state until the user presses “Resume” or “Return To Menu”. **(Medium)**
6. The user should have access to a level selection screen before gameplay starts. **(High)**
   1. Each level in this level selection will have a ‘map,’ consisting of predetermined timings and arcs for each note that gets tossed. **(High)**
   2. The maps will play alongside and relative to the timings of a specific song. **(High)**
   3. The map and corresponding song will get loaded and played after the user clicks on the level’s selection button. **(High)**
   4. Each map should at least have an easy, medium, and hard difficulty. The difference between these modes will solely be the mapping, and the song would stay the same for each. **(Low)**
7. Upon entering the program, the user must be met with a tutorial level which explains the mechanics of the game and loops these sections at least 3 times. **(Low)**
   1. To access the next section of the tutorial, the user must repeat one section correctly 3 times in a row. **(Low)**
   2. If the user fails a particular section, the loop counter gets reset. **(Low)**

# **Non-functional Requirements (10 points)**

1. The game should load quickly and not have a noticeable effect on the host operating system’s performance. **(Medium)**
2. There should be efficient handling of animations and interactions to prevent any lag or performance issues. **(Medium)**
3. The game should be compatible with all of the operating systems listed under the *Operating Environment* header below. **(High)**
4. The game should support different screen sizes and resolutions. **(Medium)**
5. The game should not crash unexpectedly. If the software does crash, the user should be offered a user-friendly error message. **(Medium)**
6. The game should protect user data and not open the user to any security risks. **(High)**
7. The volume sliders in the settings menu should not allow the user to reduce the volume below 0% or above 100%. **(Low)**
8. The game should provide varying difficulties to allow a variety of users to play it. **(Low)**

# **Use Case Diagram (10 points)**

*A diagram of a diagram

Description automatically generated*

# **Class Diagram and/or Sequence Diagrams (15 points)**

*A screenshot of a computer screen

Description automatically generated*

# **Operating Environment (5 points)**

The software will operate in any environment that meets the following requirements:

|  |  |  |
| --- | --- | --- |
| **Operating System** | **Windows** | **macOS** |
| **Operating System Version** | Windows 7 (SP1+), Windows 10, and Windows 11 | Mojave 10.14+ |
| **CPU** | x86, x64 architecture with SSE2 instruction set support. | Apple Silicon, x64 architecture with SSE2. |
| **Graphics API** | DirectX10, DirectX11, DirectX12 capable. | Metal capable Intel and AMD GPUs |
| **Additional Requirements** | Hardware vendor officially supported drivers. | Apple officially supported drivers. |

# **Assumptions and Dependencies (5 points)**

**Assumptions:**

* All team members have a basic understanding of rhythm games.
* There is sufficient technical understanding and ability to use the game engine (e.g., Unity) within the team.
* The User Interface (UI) design and gameplay functionality will be completed on time.
* Suitable music can be obtained or produced for the game without any copyright issues.
* The testing environment will not significantly differ from the actual user environment.

**Dependencies:**

* Provision of music files: The music files used in the game depend on external artists or group members, and the project depends on their progress.
* Game engine updates: The project depends on updates to the game engine used. If updates do not occur as scheduled, it can impact the project.
* Hardware performance: The game's performance depends on the user's hardware performance, and optimization work may be required.
* These assumptions and dependencies play an important role in planning and managing the project, so it is recommended to review and update them regularly as needed.