**Software Implementation and Testing Document**

**For**

**Group <7>**

Version 2.0

**Authors**:

Jake Nilsson

Patrick Canady

Estefano Cuna

Suyeol Ji

Brian Morales

# **Programming Languages (5 points)**

We are using C# for this project, since it’s the recommended language when programming for the Unity game engine. The language is used to create scripts that are attached to the game objects.

# **Platforms, APIs, Databases, and other technologies used (5 points)**

* We are using the Unity game engine and scripting API for gameplay and UI
  + Unity handles the databases for us, so we don’t use external systems for this
* FL Studio to create sound effects and music for the levels and UI
* Audacity to easily adjust sounds

# **Execution-based Functional Testing (10 points)**

* For the volume slider, we implemented sample music in the game screen and tested functionality by dragging the slider to separate values.
* For the visualizer, we used the test songs to determine what bias values would look best depending on the song.
* We built the project, navigated to the game screen, and then clicked+dragged across notes to ensure they can be cut correctly. We also dragged across notes without clicking to ensure the slices worked as intended.
* When building the project and navigating to the homescreen, we waited for visual feedback of notes being tossed up. We also included public parameters in the code that we could change during runtime for fine-tuning how high certain notes are tossed, at what speeds, etc. We would then build and run the executable to ensure the functionality is working as intended.
* Rhythm games must be suitable for players of various skill levels, making the difficulty adjustment feature crucial. Therefore, we have implemented an algorithm that adjusts the difficulty based on the player's performance before starting the game, allowing players to choose their current level. We divided this into two level selections.

# **Execution-based Non-Functional Testing (10 points)**

* Similar to the testing of functional requirements, the project is built and executed to be viewed from a user’s perspective.
* Regarding loading times and overall performance, this is quickly measured when launching and interacting with the game. With each of us running the build and having consistent results during execution, we can confidently say these non-functional requirements are being met.
* Similarly, the ability for the game to be supported on multiple operating systems and screen sizes is also tested in execution with all of our builds running on machines of varying operating systems and screen sizes.
* Regarding unexpected crashes and security, we can only test the functionality so far in execution, however both appear functional to the extent that we can execute the software.
* More tangible functionality such as the volume bar and its inability to go below 0% and above 100% are thoroughly tested and functional in execution.
* About the compatibility testing, ensures the game runs correctly across different devices and operating systems. Precise input requirements of rhythm games make diverse environment testing essential.

# **Non-Execution-based Testing (10 points)**

* Reviewed our functional and non-functional requirements to verify that they still matched our needs/goals.
* Reviewed error handling/logging in multiple classes to ensure their robustness.
* Analyzed our source code to verify that it adheres to coding best practices and is free of potential issues/bugs.
* Verified that the UI designs and layouts were consistent and accessible.