1. Introduction

1.1 Purpose

The purpose of this document is to outline and organize the requirements needed to develop our musical rhythm game.

1.2 System Overview

Our game is a fun and competitive musical rhythm game that has at least three levels which increase in difficulty as the user levels up. It should possess the ability to seamlessly interact with a database and use API calls while maintaining smooth functionality

1.3 Document Conventions

N/A (no specific document conventions)

2. Stakeholder Analysis

2.1 Identification

Customers: Individuals who will be playing/interacting with the game (mainly our teacher (Prof. Allgood), TAs, and whoever else that may be interested in playing the game)

Developers: The team responsible for system development and maintenance (Landon Jones, Alyson Mulato, Jack Sidle, Corey Turner)

2.2 Roles and Responsibilities

Customers: Play and have fun with the musical rhythm game

Developers: Design, implement, and maintain the musical rhythm game

3. Functional Requirements

3.1 System Features

Game Levels: At least three levels with increasing difficulty.

Player Progression: Advancement through levels based on performance.

Database Interaction: Seamless interaction with a database.

API Utilization: Use API calls for enhanced functionality.

3.2 Use Cases

3.2.1 Player Progression

Users progress through levels based on successful completion of challenges.

3.2.2 Database Interaction

The game retrieves and updates player data from/to the database.

3.2.3 API Utilization

API calls enhance game functionality, such as real-time score updates.

3.3 Business Rules

Players must successfully complete challenges to advance levels.

Database interactions should be secure and efficient.

API calls should be well-managed to ensure smooth gameplay.

3.4 Data Requirements

Player data: Progress, scores, achievements.

Level data: Difficulty settings, challenges.

3.5 Tech Stack

Frontend: HTML/CSS/JavaScript.

Framework: React.

Backend: Flask/Python.

Middleware: Flask Restful API.

Host/Port: Using localhost.

Data Types: Usernames, global scoreboard data, etc.

Libraries: React, Flask, SQLite, Phaser3.

4. Non-Functional Requirements

4.1 Performance

Real-time responsiveness during gameplay.

4.2 Reliability

Game uptime: 99.9%.

4.3 Security

Secure transmission of player data to and from the database.

4.4 Usability

Intuitive user interface for an enjoyable gaming experience.

4.5 Compatibility

Compatibility with various devices and platforms.

4.6 Scalability

Support for an increasing number of players and levels

5. System Constraints

5.1 Hardware

Requirements for server infrastructure to be determined.

5.2 Software

Compatibility with modern game development frameworks.

5.3 Regulatory

Compliance with relevant data protection regulations.

6. Assumptions and Dependencies

6.1 Assumptions

* Players have access to a reliable internet connection.

6.2 Dependencies

* API dependencies for real-time updates.

7. Risks and Mitigations

7.1 Project Risks

* Risk: Unforeseen challenges in API integration.
* Mitigation: Conduct thorough testing and have a contingency plan for alternative APIs.

8. Revision History

Version 1.0: Initial release [3/6/24].

Additional Information:

9. Meetings and Sprint Expectations

* Meeting Schedule: Weekly at 10 am / 1 pm on Fridays (Video conference or chat).
* Sprint Expectations: Discuss progress, address challenges, and set goals for the upcoming week during meetings. Regular communication through the chat for daily updates and issue tracking.