## Project 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 includes three “nightmare” levels which are much harder than the normal levels, being faster, having more arrows to press and having a color changer. It should possess the ability to seamlessly interact with a database and use API calls while maintaining smooth functionality.

### Requirements Gathering

Requirements were gathered through research to ensure a user-friendly and engaging gaming experience. Key requirements include multiple game levels, database integration, API utilization, and real-time responsiveness.

### Target Audience

- 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)

### Technology 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: Flask, SQLite, Phaser3

### User Management

- User Roles and Permissions
    - Regular user
        - Only able to access the levels to play
- User Registration Process
    - Common log-in is used
        - Username
        - Password

### Project Timeline

- Initial Development: March 2024 - April 2024
- Testing and Refinement: April 2024 - May 2024

### Development Environment Setup

- Install necessary software: Python, Flask
- Clone the project repository
- Set up the database and API configurations

- Run the development server for frontend and backend testing

Coding Standards and Guidelines

- Follow consistent coding practices for readability and maintainability
- Use descriptive variable and function names
- Comment code sections for better understanding

Testing Strategy

- Conduct unit testing for individual components
- Perform integration testing for system functionality
- Utilize testing frameworks for automated testing

Documentation and Knowledge Sharing

- Maintain documentation for codebase, APIs, and system architecture
- Share knowledge through regular team meetings and documentation updates

Communication Plan

- Weekly sprint meetings (video conference or chat) for progress updates and issue tracking
- Daily communication through chat for quick updates and coordination