1 Project Overview

1.1 Purpose
- The purpose of this document is to outline and organize the requirements needed to develop our musical rhythm game
- Create a fun and competitive musical rhythm game with at least three levels of increasing difficulty
- Seamlessly interact with a database and use API calls for enhanced functionality

1.3 Target Audience or Users
- Customers: Individuals who will play and interact with the game, including our teacher (Prof. Allgood), TAs, and other interested players

2. Project Scope

2.1 Features and Functionalities to be Included
- Game Levels: Three levels with increasing difficulty
- Player Progression: Advancement based on performance
- Database Interaction: Seamless interaction with a database for player data
- API Utilization: Use API calls for real-time updates and enhanced functionality

2.2 Exclusions and Limitations
- Limitations on the number of levels initially included

3. Requirements Gathering

3.1 Stakeholder Interviews or Meetings
- Conduct meetings with stakeholders to gather requirements and feedback

3.2 Documentation of Functional and Non-functional Requirements
- Document both functional requirements (e.g., game features) and non-functional requirements (e.g., performance, security)

3.3 Prioritization of Requirements
- Prioritize requirements based on importance and feasibility

4. Technology Stack

4.1 Programming Languages and Frameworks
- Frontend: HTML/CSS/JavaScript
- Framework: React for dynamic UI
- Backend: Flask/Python for server-side logic
- Middleware: Flask Restful API for database interaction and API calls

4.2 Databases and Data Storage Solutions
- SQLite for local development and testing
- Consideration of scalable databases for production

4.3 Third-party Libraries or Tools
- Use of libraries like React, Flask, and Phaser3 for game development

5. Project Timeline

5.1 Milestones and Deliverables
- Milestone 1: Initial game prototype development
- Milestone 2: Integration of database and API functionalities
- Milestone 3: Testing and debugging phase

5.2 Estimated Timeframes for Each Phase
- Planning: 4 weeks
- Development: 8 weeks
- Testing: 2 weeks

5.3 Dependencies Between Tasks
- Development depends on successful planning and requirement gathering phases

6. Team Roles and Responsibilities

6.1 Roles Within the Development Team
- Developers: Landon Jones, Jack Sidle, Corey Turner
- Project Manager: Alyson Mulato

6.2 Responsibilities and Tasks for Each Role
- Developers:
    - Landon Jones
        - Game logic and game loop
        - Creation and modification of sprites
        - Audio programming and audio syncing
        - Gameplay development
        - Level design
        - Initial backend design
    - Jack Sidle
        - Implemented login functionality with the database, making it so that you can only have one account associated with a name
        - Added level scores to the DB associated with each level
        - Update users completed levels in DB to only allow them to do the levels they have unlocked
        - Keep track of total scores to later be sent to the top score list.
    - Corey Turner
        - Frontend aesthetics
        - Formatting the level select page and login pages.
- Project Manager: Coordinate tasks, schedule meetings, and track progress

6.3 Communication and Collaboration Channels
- Regular team meetings and communication through chat or video conferencing tools

7. Development Environment Setup

7.1 Version Control System

- Use Git for version control and collaboration

7.2 Development IDE or Editor

- IDEs like Visual Studio Code for frontend and backend development

7.3 Testing Frameworks and Tools

- Postman to test our API
- Werkzeug Debugging for Flask Errors

8. Coding Standards and Guidelines

8.1 Coding Style

- Followed standard coding convention and syntax for each language used
    - Prioritized following this standard to decrease the level errors and improved readability

8.2 Naming Conventions

- Use descriptive and consistent naming conventions for variables, functions, and classes

8.3 Documentation Practices

- Write a comprehensive README.md for project setup and usage instructions

9. Testing Strategy

9.1 Types of Testing

- Unit testing for individual components
- Integration testing for database and API interactions
- End-to-end testing for overall functionality

9.2 Test Plan Creation

- Develop a test plan outlining test cases and scenarios

10. Risk Management

10.1 Identification of Potential Risks

- Unforeseen challenges in API integration
- Performance issues with real-time updates

10.2 Risk Mitigation Strategies

- Thorough testing and integration testing for APIs
- Monitoring and performance optimization for real-time functionalities

10.3 Contingency Plans for Critical Risks

- Have alternative solutions ready for critical functionalities

11. Documentation and Knowledge Sharing

11.1 Documentation of Project Architecture

- Document the project architecture, design decisions, and component interactions

11.2 Knowledge Sharing Sessions

- Conduct knowledge sharing sessions within the team for code reviews and best practices (usually done during SPRINT sessions)

12. Communication Plan

12.1 Communication Protocols

- Regular team meetings for status updates and progress tracking
- Use of chat or messaging tools for daily communication and issue tracking

12.2 Reporting Mechanisms

- Prepare progress reports and status updates for stakeholders
- Use issue tracking tools for bug reporting and resolution

12.3 Stakeholder Communication Strategy

- Regular updates for stakeholders to gather feedback and input through SPRINT submissions

Additional Information:

13. Meetings and Sprint Expectations

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