**Rhythm**

**Runner**

**Project Report**

California State University, Northridge

COMP 467 Multimedia Systems

Fall 2017

**Prepare by Team X:**

Dillon Hour

Hongyou Xiong

Paul Fussell

December 03, 2017

**1. Project Goals**

Games that have been developed in the past have mainly focused on creating visual interactions between the players rather than audio interactions. Rhythm Runner, a 2D side scrolling runner game, will utilize features of the music to create the gameplay experience. Leveraging what we’ve learned in this class, the goal is to utilize sound manipulation techniques involving fast fourier transforms to create an interactive experience that derives from the music playing. We aim to have a fully finished prototype for a game by the end of the semester. We will be using Unity as our main game developing platform which will allow us to create a 2D fully functional game. We will be designing our own web based game, that combines both audio and visual interactions between the players.

In order to make sure we are able to learn the framework that Unity provides as well as deeper understanding of C#, we have planned out how to best scope our work so that there’s a playable functional prototype by the end of the semester. This will include a minimal, but functional UI (main menu, level selector, help menu, score, and pause menu); it will also utilize some of the sound libraries that Unity provides to create levels based off the audio as well as adjust the music playing depending on how the player is doing. As it is a prototype, the goal is not to make a fun game, but to leverage audio manipulation to create interesting levels.

**2. Project Description**

**2.1 Project Overview**

Rhythm Runner is an independent and self-contained product for COMP 467 Multimedia Systems. This product is under development by a team of three members. The goal is to complete this project within a nine-week time span. Developing this game will provide a great opportunity for each member to be familiarized with C#, Unity and audio manipulation algorithms involving FFT.

Rhythm Runner is an arcade style 2D rhythm game which allows the player to take control and run through obstacles. This single player fast action paced game will be comprised of multiple levels and different stage obstacles. The user’s goal is to reach the finish line without getting destroyed and to collect as many coins as possible.

Rhythm Runner will be compatible on Windows, and Mac OS as well as working with all major web browsers such as Chrome, Firefox, Internet Explorer, and ect..

**2.3 Project Scope**

Rhythm Runner is a side scroller rhythm running style game. The purpose of this game is combine musical beats, and sound frequencies with the player actions. Each player will have a one to two minute experience on each level with 3 different levels to complete. The player's objective is to run across platforms, collect coins, and dodge the pitfalls and traps along the way. The coins collected will gradually increase the pace of the music as well as the speed of the character, making the game feel more hectic as the player does well.

**3. Project Breakdown**

**3.1 Tools**

**Game Creation** - Unity, Visual Studio, and Photoshop.

**Communication** **/ Source Control** - GitHub, Discord, Gmail, Google Drive, and Skype.

We’ve chosen to use Unity for this project as the game engine of choice. To write C#, we’re going to use Visual Studio 2017 as an IDE and Photoshop for basic 2d sprite manipulation. To communicate and maintain work, we’re going to utilize Github for source control and Skype / Discord for communications. Documentation will exist and be shared across Google Docs, so that we can collaborate on work.

We looked at a bunch of options for the game engine: Ogre, GameMaker, Unity, and Unreal Engine. Ogre is an open-source game engine that does provide a lot of useful features. The language it supports is C++, but it does not provide any physics or audio capabilities. That makes it unusable for the project even though it is the main open source option. We could import an audio library but it would introduce more risk into the project.

GameMaker is a 2D visual programming IDE to generate 2-dimensional games. It is considerably slower to run than C++ or C# because it is a scripting language engine. In addition, it is hard to write the algorithms necessary to adjust and manipulate the audio because it is all visual scripting. GameMaker is also a proprietary license that you can try for free but is not free to use. This makes it more difficult to choose as an engine.

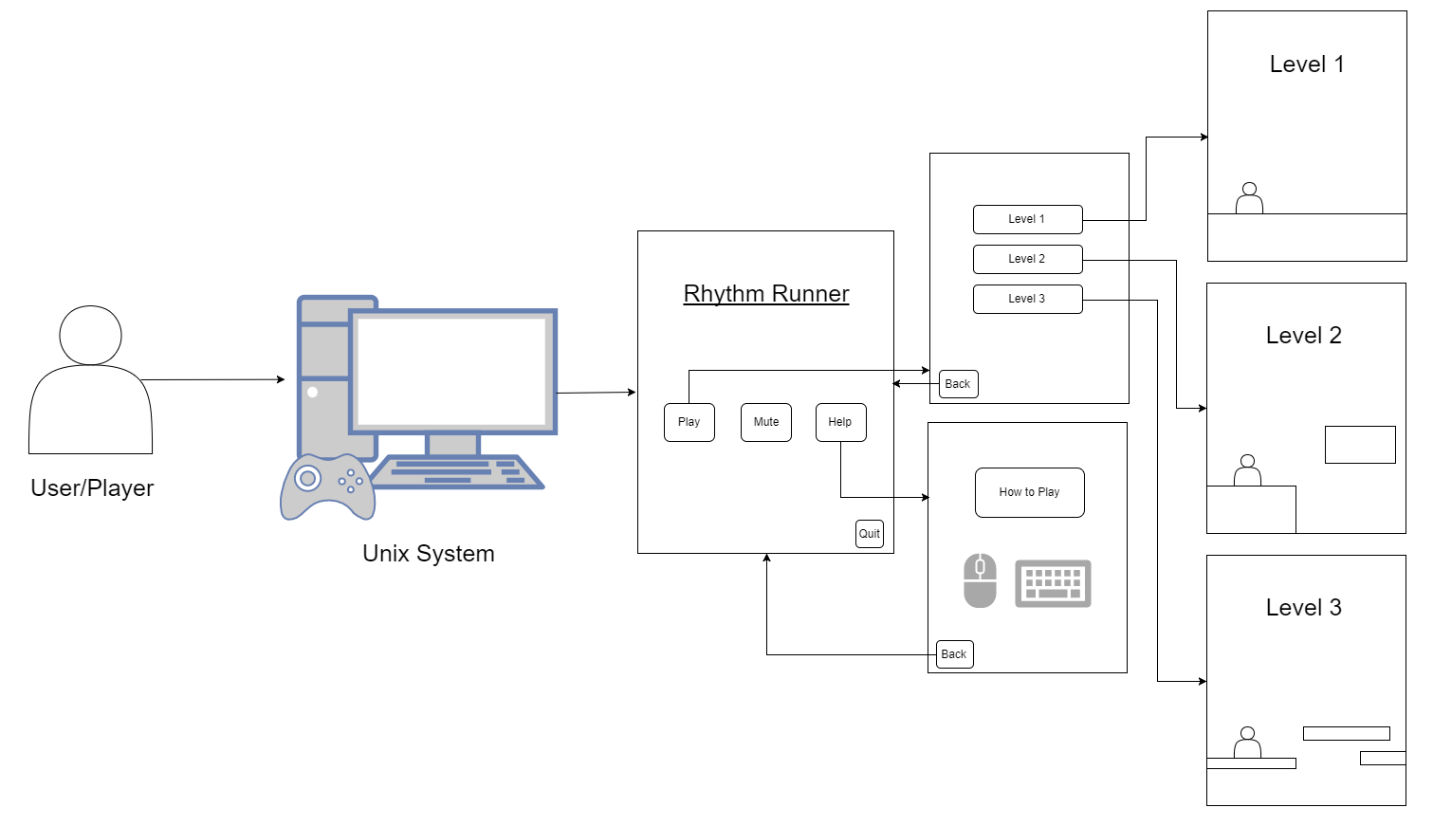
Unity and Unreal stood out as the two enticing choices; both provide deep tutorial libraries and also a wealth of features. Unity uses a mixture of C# and visual scripting while Unreal uses C++ and Blueprint, their own version of visual scripting. They both support audio manipulation through an API and are typically used in industry setting. Both have lots of support online and are constantly being worked on, and are free for users to use as long as the projects are below a certain profit line. We opted to use Unity because our team has used it before and tends to be easier to use for simpler 2d games than Unreal.

We pretty much unilaterally chose to use Visual Studio for our choice of IDE because of how many features is built into it. It comes with a compiler for various languages including C#, and also a debugger that bare-bones IDEs do not provide. There are also GitHub integration plugins in the Visual Studio package library, which helps to reduce the friction between source control.

Communications are going to be over Skype, Discord and text message. This allows us to work even though we are all scattered about and communicate in real time with one another. Files that we collaborate on will be housed on Github for code and Google Drive for other documents. Google drive is convenient for collaborative sharing and editing of files. The other options included Dropbox and Microsoft One, but there were no competitive advantages to use one or the other and we’ve used Google Drive primarily in the past, so we chose to use that instead of others. Github is more convenient for sharing private repositories, especially for students. It is easy to integrate with Visual Studio and serves its job well for code files. While Git is not great for binary assets, it will be sufficient as the scope of the project itself limits the amount of assets needed in the system.

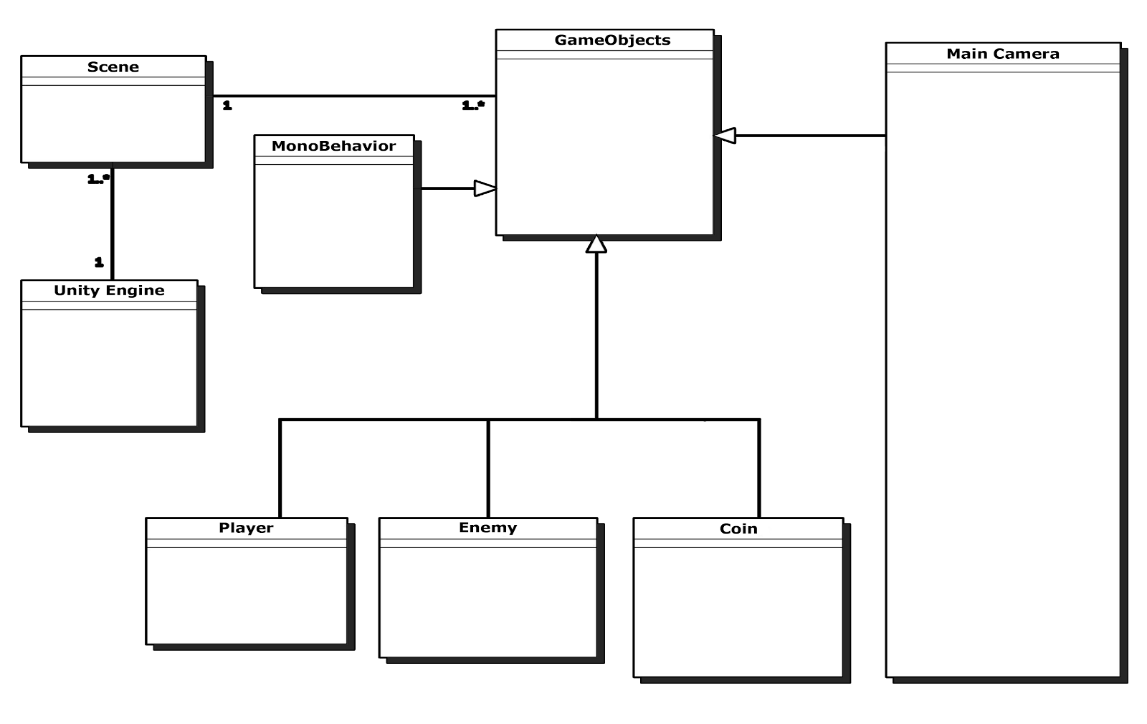
**4. User Specifications  
4.1 Functional Requirements  
4.1.1 User Interface**  
Rhythm Runner will be supported on the web and will also be downloadable for any player who wants to play directly off their computers instead. We will ensure that the user interface will easy and simple to use. Once the player starts Rhythm Runner, he will prompted to the main menu where he/she will have the options to; play, mute, ask for help, or quit the game. If the player presses help, they are taken to the level select screen where the player may pick a level to begin playing. If the player presses mute, the menu audio will be silenced. If the player presses help, the player will be prompted with controls and instructions on how to play the game. Lastly, if the player wants to quit the game, he/she will have the option to quit in the main menu.   
  
As the user is playing the game, there will be a ‘pause menu’ user interfaces that will be included. When the game is paused, the user will be able to go back to main menu, quit the game, or restart the level. If the player gets killed during the level, he/she will have the respawn at the beginning of that level.   
  
**4.1.2 Hardware Interface**  
The user must have a fully functional computer with the Mac or Windows operating system, as well as have internet access. Our game will be web-based, so the player needs to be able to access the web to play the game. Additionally, the user must have a keyboard or mouse to be able to control the player's movement in the game. Optional hardware may include speakers that will allow the user to hear the sounds and music within the game.  
  
**4.1.3 Software Interface**  
We will build our game using Unity’s WebGL’s build option which will allow the us to make it playable through the web. WebGL is a Web Graphics Library built on the JavaScript API used to render interactive objects that include both 3D and 2D graphics on compatible web browsers such as Mozilla Firefox or Google Chrome. This makes it so the user does not require any plug ins for the browser because it makes use of the HTML5 elements and OpenGL ES 2.0.  By using an existing and proven library for the basis of the game we can ensure that time is spent more on gameplay development and that Rhythm Runner has the reliability that gamers today demand. The web browsers that we intended to build it on are Google Chrome, Firefox, and Safari.  
  
**4.1.4 Communication Interfaces**  
There will be no communications interfaces since this will not be a multiplayer, or online game that will allow player to player interaction. All data from the game will be save locally onto the user's computer.   
  
**4.2 Non-Functional Requirements  
4.2.1 Performance Requirements**  
The game should run at least 30 frames per second and use at most 512MB of RAM. Dedicated graphics card should not be required to play such as those high end graphics cards such as NVIDIA, or AMD Radeon graphics. This game will be playable with any sound card capable of stereo output.  
  
**4.2.2 Usability Requirements**  
Novice and inexperienced game players should be able to play our game. There should be little to no learning curve for all players. The game should have straight forward gameplay and basic/intuitive mechanics. Once the player starts gameplay, it will be as simple as to click and jump.   
 **4.2.3 Security Requirements**  
There will be no security requirements since the game will be played locally and will be saved onto the user's workstation. The only thing that can be stolen is the game data, but no personal information will even be required from the user.  
  
**4.2.4 Maintenance Requirements**

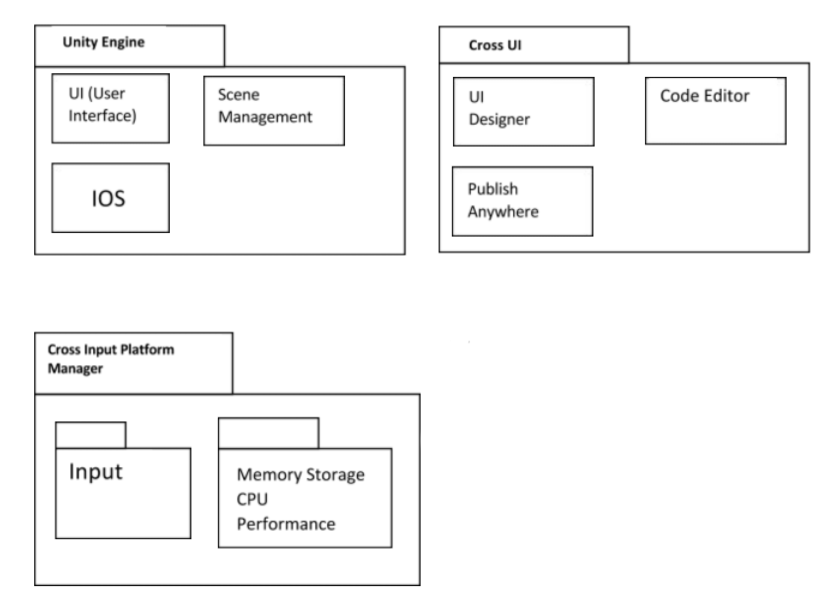
Assets of each team member should be separated so that everyone’s contribution may be maintained separately. Additionally, unused assets will be deleted in order to keep the memory of the game low and efficient. Therefore, by delete these unused asset, it will create a faster and more optimized game. Future patches will only address game-breaking bugs after release. Beyond that, the team will be continuously test the games for any further bugs and will take action upon each bug that is encountered.   
 **4.2.5 Extensibility Requirements**  
When the semester is finished, we plan on extending the game by creating more levels and releasing new characters. Therefore, we will be expanding the gameplay by adding new player objectives and goals. However, we will still implement the same gameplay style of having the players actions correlate with the sounds in the game.

**4.3 Wireframes**

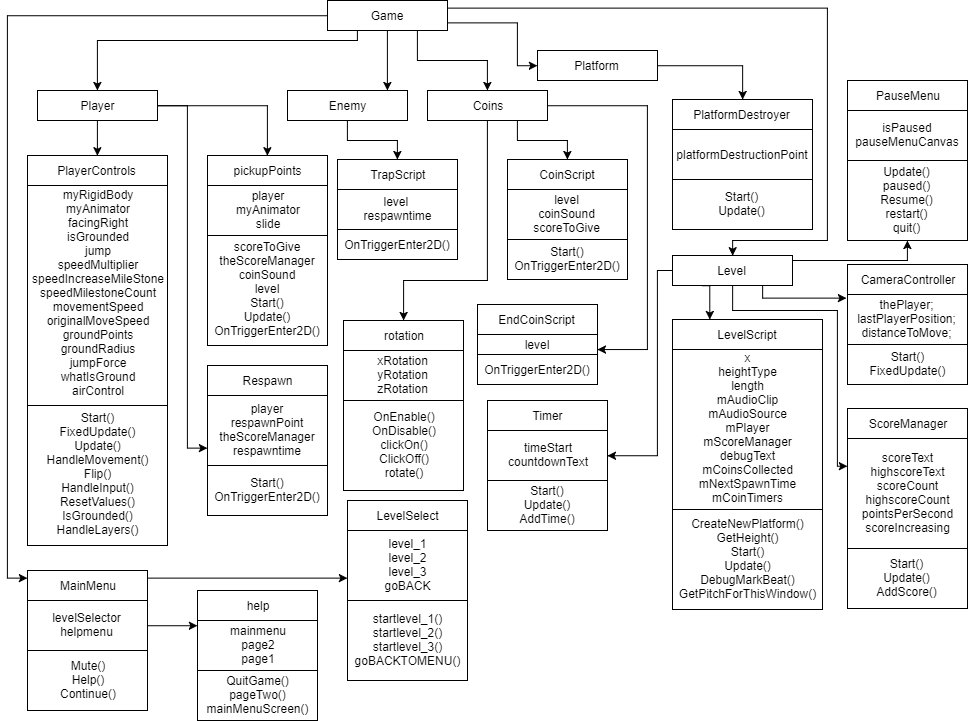
**5. UML Diagrams**

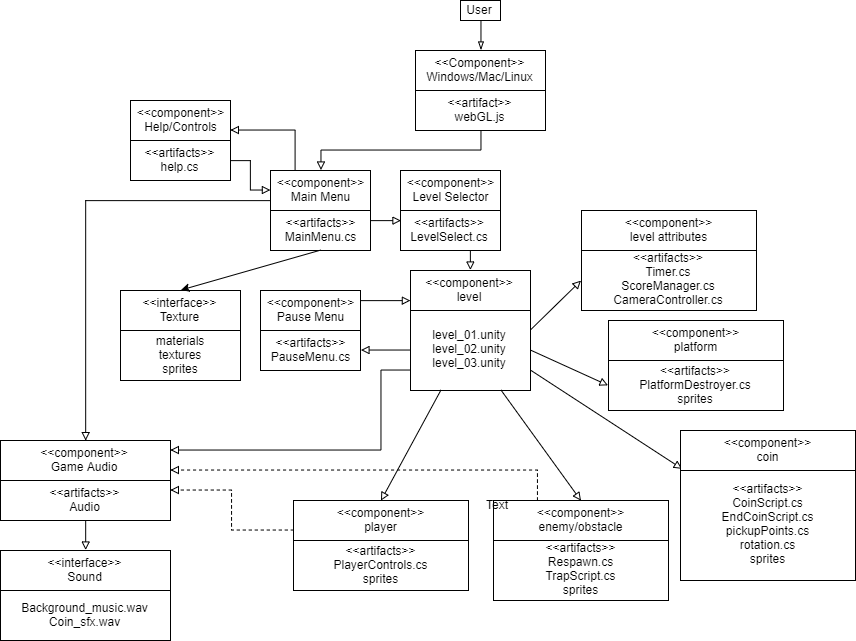
* **Class Diagram -**

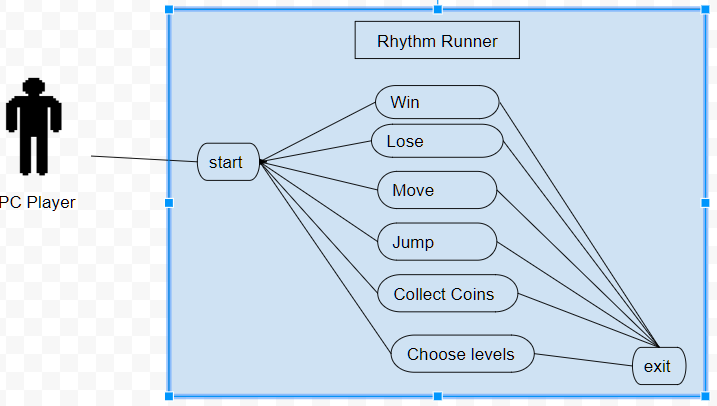


* **Package Diagram** -

* **Object Diagram -**



* **Component Diagram -**
* **Use Case Diagram -**



**6. Code Guidelines**

**6.1 Naming Conventions**

Class names will be determined and named according to the functions of the script. For example, if the script deals with the movement of the player, then we will name the class PlayerController or if the the script deals with the score, then we will name it ScoreCounter. Therefore, our classes will be named according to what they do in the game.

Variable names will be lowercase, short, and accurately describe the values that are stored within them, as well as their purpose. For example, we will have a function that regulates the players movements and we will call name the variable speed. Speed will allow us to change how fast the player moves and runs.

Functions will be named according to it specific actions towards the game, such as start(), or update(). Start() will be code that needs to run when the game starts and Update() will run continuously until the game ends. There will be more functions that will be called upon, but refer to our architecture to see the list of functions we will be using.

The asset will be named accordingly depending on the character type, music, or to what they are related to. There will be only one character and it will be named “Player”. Objects will be named to what they are assigned to do or be, such as “ground”. Ground will be an asset that the user can use to create the floor layout of the game. The reason for naming the object this way, is to ensure the developer understands what they are used for.

**6.2 Paradigms**

We will be using a mixture of a bunch of simple paradigms. The core game will be a state machine, where each phase of the game is a different state. Breaking the game into Start Screen, Game, Paused, and Quit will help make the game much more modular and each state can manage its own data.

The objects and character will be objects each housing components that are responsible for rendering, audio, I/O, and others. Component based objects mean that the object is composed using various components that allow for modular behavior. This will help us be able to work in parallel (For example, one person working on the input for the character and another person working on the animation or rendering will be able to work in parallel because they touch the character itself minimally.

**6.4 Frameworks**

We will be using all of Unity’s built in frameworks and APIs, which include their own C# scripting languages and we will be using Unity’s collaboration features. The collaboration feature allows developers to interact with other developers which allows them to share code and work on the same project.

**6.7 Security Guidelines**

There are no security guidelines as there are no security requirements. We are not implementing any user account information. Therefore, we will not have a need for security for our application. However, we will be using the built in Unity Collab feature to ensure that our code and development progress is saved on onto the cloud.

**6.8 Constraints**

**6.8.1 Users**

The user must have full internet access so that they can play the game since it will be web based. If the user does not have internet, then he/she cannot play the game since it will be on a website. Additionally, to prevent any lags during gameplay, it is suggested that the user has a computer that was built between 2014 - present.

**6.8.2 Hardware**

For hardware, the game requires at least 512 mb of ram, a monitor, a keyboard or mouse, 1 gb of free space and an integrated graphics card. Without these hardwares, user may be limited or may not be able to play the game.

**6.8.3 Software**

The game is constrained to run on unix only systems and only mainstream browsers. In other words, the software can only run on Windows, Mac, and Linux operating systems and in Chrome, Firefox and Safari browsers.

**6.9 Usability Guidelines**

**6.9.1 Menu Organization**

We will create the main menu really basic and easy to use. The main menu will consist of play game, mute, and help. Each function will navigate the player to another screen. Play game will bring you to a menu with a list of levels to play. Mute will mute the game. Help will give the player instruction on how to play.

**6.9.2 Buttons Used**

With the keyboard and mouse, the player will press space bar or click the left click button on the mouse to jump.

**6.10 Links**

* <https://unity3d.com/>
* [https://github.com/powerfuldil/DH-PF-HX-COMP47-RhythmRunner](https://github.com/powerfuldil/DH-PF-HX-COMP47-RhythmRunner/invitations)
* <https://www.google.com/drive/>
* <https://discordapp.com/>