Radio Rideshare – Literature Review and Project Plan

Abstract: Our goal is simple: create a fun, enjoyable rhythm game with a custom level editor. However, there are many crossroads to be met which we have considered during the design and formation of our project that must be addressed before we scaffold up and build upon our original idea. Planning for the future is key here. Every choice we make now regarding structure and foundation sets the rulebook for any future progress in this project. Over the course of this paper, we intend to lay out the key points we have had to consider so far, along with they way we chose to go as well as why. These issues range from abstract topics such as our music and art style to more technical conversations such as what engine to use and what language to program in.

I. Engines and Languages

Game Engine

When designing a game, no matter how basic, one must face the question of what engine to use. The obvious choices that we came to were Godot, a completely free and open-source 2D and 3D game engine, and Unity, a (mostly, as long as you don't get popular enough) free closed-source 2D/3D engine. The differences in usage are hard to compare – while Godot is much newer and therefore has less documentation on it, Unity is a much heftier engine to work with.

In the end, we opted for Godot, for a few reasons. First, Godot boasts an extremely relaxing file size around 130 megabytes, which is completely dwarfed – almost laughably so – by Unity's 5 gigabyte minimum size, with required downloads and installations sometimes taking up closer to 100 gigabytes. Above many requirements, we wanted to make this game portable. This would

not be a guarantee if we relied on Unity. The other main reason we choice Godot is for its custom scripting language, GDScript. Godot's custom language for game design is incredibly simple to read and understand, and the documentation on it is thorough and clear. This will be extremely beneficial to us as we develop the game.

Programming Language

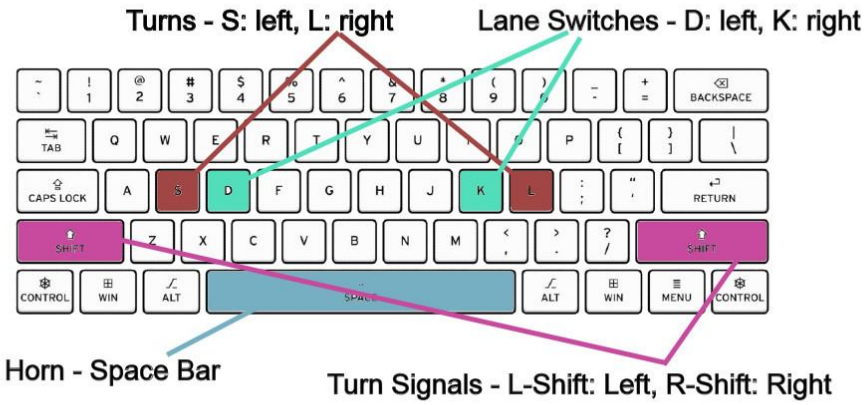
The next topic of immediate concern is language. After settling with Godot, our next conversation focused around how to make best use of it. The two languages of choice we have are C# and GDScript. We intend to make various use of each, depending on what is required. GDScript was built to work well with Godot, with C# support being a much newer addition. As such, the GDScript support is much more thorough, and some C# integration and code will require a bit more work. However, the sheer difference in maturity between C# and GDScript makes it a difficult conversation. Should we opt for GDScript and its integration, or C# and its libraries and history of documentation and previous user-discussions online? The answer is simple, both. Godot allows for integration between C# and GDScript across a project. Unfortunately, it does not work on the same file, but the project itself allows interaction between components built with GDScript with those built with C#. This gives us the freedom to employ certain aspects with GDScript when we find that the integration is more important, while other parts of the project that may require C#'s comprehensive libraries can be done in C# without issue.

II. Gameplay

Controls

Having settled the various “how’s” of our project, we can begin discussing the “what’s”. That begins with control scheme. First thing to note is that we intend to make this a strictly keyboard-and-mouse game. While later down the line, we may consider expanding towards Controller, the intention is to begin and focus on *Radio Rideshare* as a PC game. Traditionally, PC games control with a “W, A, S, D” scheme by default. This would be easy for even a slightly experienced gamer to pick up instantly. However, it also limits the controls of our game and reduces our goal of an imitative control scheme. Rhythm games often do this, employing unique control schemes to fit their needs. Examples of this include *Trombone Champ*’s sliding mouse control, *Osu!*’s mouse-and-click scheme (often played with a pen and tablet), and *Guitar Hero*’s famous custom controller.

Our options are to either stick with a classic scheme, either “W, A, S, D” (or some simple variation), or a custom control that does what we hope it to – make the controls an extension of the immersive experience. The goal for a good set of controls is to imitate the vague feeling of driving a car – as much as possible without using a wheel. We plan to take the following aspects of a car and bind them to keystrokes: left blinker, right blinker, left turn, right turn, left lane switch, right lane switch, horn. Keeping in mind that we want the game to be enjoyable and easy to pick up, not tedious and confusing, we must balance our hopes for a perfect control scheme with something that is digestible. A possible solution is to directly map the controls as a symmetrical pattern with the L-Shift, R-Shift, S, D, K, L, and space bar keys, respectively. See the diagram below for clarification and note how the layout is meant to mimic the actual sides of the car the player affects when controlling the game.



We believe that with this scheme, the player doesn't sacrifice gameplay potential, while giving us the ability to throw a unique twist on a control scheme that gives the player a sense of meaning to the keys. Had we gone with a different scheme, we would have had to consider our desired controls, since having W, A, S, and D control left and right signals as well as turns would result in a confusing mess of an experience.

Since we opted for a symmetrical layout, we also gain the ability to easily teach the controls. It is standard for games to have tutorial sections, and since our controls are based on symmetrical actions performed with each hand, we essentially turn our seven-key control scheme into a four-key scheme, extremely simple and easy to package for the user.

Gameplay Loop

As with any game, our mission is to create a satisfying gameplay loop that keeps the user involved. Gameplay loops are abstract concepts, ranging from "beat world levels, fight a boss,

move on to next world", to "perform a song to collect currency, use currency to better equipment and get new songs, repeat". Here, since rhythm games are based inherently on the music they go along with, we are limited by how much music we can produce for the game. We plan to have a simple story mode – a set of songs to serve as the "official" campaign of the game, created and implemented directly by us. However, the main gameplay loop we hope to achieve is through user-created levels. Community-backed games that allow for custom-created levels open the door to infinite possibilities, as well as creating a secondary user-pool.

With traditional games, you have only "players", who can only ingest whatever levels are created by the original makers. In community-sourced games such as *Osu!, Super Mario Maker,* and *Clone Hero*, the bulk of the game is user-created levels. This not only expands the available level options to players, but also creates another source of users, the "creators". The community grows in two directions, often with users being both players and creators. The added creative factor becomes a game unto itself, which is what we hope to achieve with our level editor, which we consider as a secondary gameplay loop beyond the campaign.

III. Art and Styling

Style

Any game built must address how it wants to package and represent itself. What art style we choose should reflect the tone we wish to portray and the aesthetic we aim for. There was not much debate here. Our team agreed quickly that we want the game to be made in a cartoon style. The freedom of creativity, the large base of resources and models that already exist, and the connection we have to artists who are willing to work with us to create the deliverables made this

the obvious choice. There is no “correct” way to go about an artistic style, and someone familiar with rhythm games could point out the range of styles to accompany the gameplay. For example, *Rhythm Heaven* uses a unique 2-D comic art style, *Guitar Hero* a style that attempts realism, *Rhythm Doctor* a pixelated art style. It really is up to us with which direction we want to go in creatively, and as a team we decided that our creative choice was for 3-D voxel graphics. That means pixel graphics in 3-D with perfect cubes.

IV. Music

Composition

We want to create our own music for the game, but that is an extremely time-consuming process. Trying to create a long game with a custom soundtrack will be a difficult process, especially if we want the music to be of good quality. Therefore, we will limit our own music to what we feel confident with and allow the community-created levels to appear in a way that gives players an easy access to them. This easy access could be in a database or stored on local files the user can navigate to and curate themselves. The main idea here is to not worry ourselves about being musical maestros (potentially derailing our original intentions with the project as a whole) and instead focus on a few core pieces as our proof-of-concept and campaign backing while otherwise emphasizing the gameplay and functionality.