COS426 Written Intermediary Report

Tap Tap

Daniel Lee and Alex Dipasupil

I. Abstract

The goal of this project was to create a rhythm game, using Three.JS as the framework. We create falling notes that approach a grid of player target boxes, and in time with music, the player is expected to hit keys to accurately clear notes. With a life system, the player is incentivized to hit notes correctly, and at the end, they are able to see how well they performed with a score total.

II. Introduction

The goal of our project was to create a rhythm game in JavaScript. Using Three.JS graphical components to display falling notes onto a grid, we play music and task the player with correctly timing the falling notes with the music on the correct locations in the grid. As this is a game, it is targeted towards people who typically play rhythm games; on a broader level, however, it can potentially aid with mastery of fine motor control and reaction speed.

Rhythm games are a niche genre of gaming, but there is nevertheless a great body of previous work. One of the most popular rhythm games currently released, played by millions, is Osu!, a game in which circles appear on the screen, and the player is tasked with clicking these circles with the appropriate timing in the correct locations. There are thousands of player-made maps, all of which feature unique timings customized to fit each individual song. One other popular rhythm game is Dance Dance Revolution, frequently featured in arcades. Music is timed to falling arrows on the screen, and these arrows correspond to four large touch-sensitive pads on the ground. The player is meant to hit these pads with their feet, and more difficult songs lead to rather frenetic leg movements. Some installations and variants of the game feature calorie counts after a song is completed, doubling as a type of exercise machine.

There are many other rhythm games as well, made by varying developers for many different platforms, ranging from arcade machines (Dance Dance Revolution and its numerous variants), computers (Osu, Stepmania, Friday Night Funkin), mobile devices (Deemo, Voez, Lanota, Arcaea), consoles (Guitar Hero, Rock Band), and many others. Ultimately, many different concepts for rhythm games have been made, and each one offers a unique spin on the genre; however, the one common feature that all these games share is the concept of timing some sort of button press to music. (As this is the core concept of a rhythm game, this is always held constant.) The unique spin that we offer is having a three-dimensional graphical format; nearly all of these games present notes in a two-dimensional space. Using Three.JS, we display three

dimensional notes that approach the grid, and in each lane, the player is expected to align these notes with the receptacles that correspond to input keys.

The approach we took was to create objects that correspond to music notes and then to assign them to lanes depending on the music. Depending on which of the five lanes the note falls into, it is assigned a color. Then, at a preset speed, the note falls down into the lane. Each lane contains a box that corresponds to the player's goal; with each key corresponding to a certain keypress, the player is expected to time their keypress to the falling note, pressing the key when the note is right above the box. We think this approach works well for the case that music is not overly complicated; when the piece is too fast, we risk having too many notes for the player to handle, and notes might appear in too many lanes at the same time.

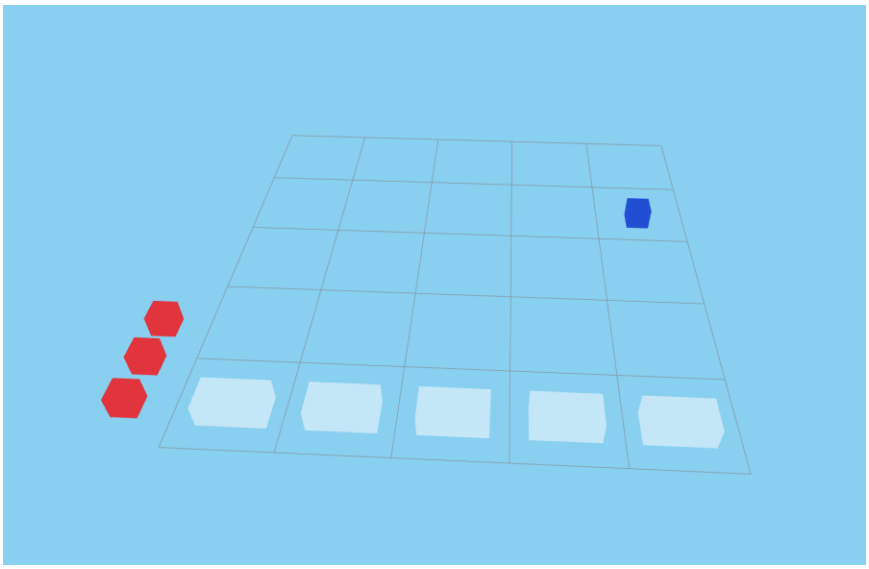
III. Methodology

To execute this approach, the first thing we needed to do was create a music note object. This music note uses Three.JS to take the form of a box, and as mentioned earlier, its color is assigned depending on which of the lanes it was randomly selected to enter. Aside from the music note, we also feature a score element that tracks how many music notes the player managed to successfully time; this number goes up for each consecutive note timed correctly. There is also a life system, in which the player has three lives; each missed note subtracts one life from the total, and once the player hits zero lives, the game ends. For the minimum viable product, we wanted to make sure that this core system worked; adding music was the secondary component to the game's functionality (although in a finished product, it is critical). If the player wants to, they are able to drag the camera around, changing the orientation of the player board. This can introduce interesting ways to play the game, using different view angles to change the direction in which notes fall down.

One of the pieces we chose not to implement is an audio parser of any sort. There are some rhythm games that feature automated audio parsing, allowing the user to input a song of their choice and have the game automatically output a customized, specific map for the song in question. We believe that this is outside the scope of this project and focusing too much on one particular component of making the rhythm game work would overall negatively impact other, more important features of the project. As a result, the current MVP implementation ignores aspects of the audio meant to be played along with the game, and simply focuses on the purely gameplay aspect of hitting notes into their correct lanes.

IV. Results

We measured success simply by playing the game. The final score at the end was able to reflect how many notes were hit, and for each missed note, a life was subtracted from the total. With the life counters displaying on the side of the screen as red boxes, and the notes falling down as various multicolored boxes, the player is able to visually see how the game plays out;



since each colored note corresponds to a specific lane, the player is able to mentally and permanently associate a color with a specific lane.

V. Discussion

Overall, the approach we took is promising. Although the game board displays as a grid, as soon as notes drop, we believe it is clear how each lane functions. One potential variant of the approach we took that could have been more effective is to introduce some sort of internal timer that ticks regardless of the player's frame rate; as it currently stands, as with most Three.JS programs, the speed of the program depends strictly on the player's framerate. This could lead to potential issues; if a player with a faster computer plays the game, they might have a drastically different experience from someone who plays on a slower computer.

The most immediate follow-up work is to now add music to the game. Having done the major gameplay components, since this is a rhythm game, we need to insert appropriate music that allows the user to interact with the game in time with the music. After adding this, the two components of gameplay and audio will be able to go together, creating the full experience of a rhythm game. By doing this project, we learned in greater detail how Three.JS works with random objects, and we were able to use what we learned about object collision earlier in the course to properly deal with accurate note timing.

VI. Conclusion

We were able to reach our goal of the minimum viable product. The core component of the game itself functions as intended, and although the appearance is relatively barebones, we were able to use Three.JS to display all the most core components. Naturally, to move forward from our MVP, we have many features to work on next.

One such feature would be a more robust scoring system. As it currently is, the game features a highly binary system; if the player hits the note, they receive a point, and if they miss the note, they lose a life. Most rhythm games feature a less granular system; depending on precisely how accurate the player's timing is, they receive more or fewer points. For example, in one game, players receive 300 points for hitting a note within 50 milliseconds of the intended time, 100 points for hitting it within 100 milliseconds, 50 points for hitting it within 200

milliseconds, and zero points (as well as health point loss) if they are outside of this 200 millisecond window. This incentivizes the player to not only hit the notes, but to time it very carefully; the intention is that the notes are tightly timed to the music, so the player is able to hit their notes in line with what they are hearing. As alluded to earlier, many rhythm games also feature a health point system instead of a life system. Health points start at maximum; incorrect or badly timed notes subtract a large amount of health, and correct notes add a tiny bit of health back. In this way, even if the player makes a mistake or two, if they maintain their timing and hit other notes correctly, it is possible to recover. Another feature of scoring that other games feature is a combo system; for hitting many notes correctly consecutively, each successive note grants increasing numbers of points. In this way, the player is incentivized to maintain long strings of correct notes, and they are thus rewarded for maintaining them. Popular rhythm game terminology is a "full combo" for completing an entire song without missing a single note, and "full perfect" for completing an entire song without mistiming a single note at all. One other potential spin on the scoring system is to add different score amounts for different note types; although it is difficult to feature notes such as slides when the method of control is a keyboard, adding some increased score variance would introduce additional complexity. There are very many ways to improve and iterate on the baseline scoring system to increase player interactivity, and these are just a few of these ideas.

One other aspect of the game to focus on would be the graphical aspect. As it currently is, there is very little visual fanfare for hitting a note correctly apart from the note disappearing, and the only visual notification that a note was missed is the removal of one of the life counters on the side of the screen. (There is also no auditory cue at all for a note being hit correctly.) One of the first things to do would be to create some sort of effect for a correct note; this could take the form of a series of particles emanating from the player's target box, or perhaps the target box lighting up and changing color. In this way, the player would receive more positive visual feedback for hitting a note correctly, adding an incentive of sorts for playing well. Conversely, we could also add an increased impact when a note is missed; perhaps the screen could change color, or the life icon could have an effect that pops more when it expires. Additionally, we could make the background much more interesting; perhaps some sort of moving geometric plane, or a background scene that changes depending on the music would add extra visual flair. Another change could be to make the music notes themselves more interesting; instead of being just a box, they could take some sort of musical note shape, or perhaps the color of the note could change depending on more than just the lane that the note is in.

Lastly, one other major improvement to make would be to vary the gameplay more radically depending on the music. Though most traditional audio formats feature no information about music notes in them, one music format that does is the MIDI format. This format, tailored for music, features notes as a data structure (instead of audio waves), meaning that notes can be parsed on an individual basis. This would make it possible to have note presses be automatically tailored to precisely fit the music in question. However, implementing this would be rather

involved; not only would the program have to feature a MIDI parser, but it would also have to include a MIDI synthesizer in order to output audio for the player. Making all these different components work together would take many steps; one potential simplification would be to include a feature that simply changes note speed based on a supplied factor. This could be tailored to a piece's tempo, but it could also alternately function simply as a difficulty meter, letting the user adjust how quickly they wish for notes to approach.

One of the primary issues to revisit is accurate note collision detection. As it is, due to our implementation, it is not completely clear how close the player is to hitting the note correctly. Of course, the primary issue to address is the note collision itself, but to communicate the player's accuracy (or lack thereof) to them better, one of the avenues to tackle would be improved score calculation, as described above.

Additionally, general graphical cleanup would help as well; currently, the controls box in the upper right corner does not function as intended, although it is clickable. Letting the player interact with this in a more meaningful way would be much clearer.