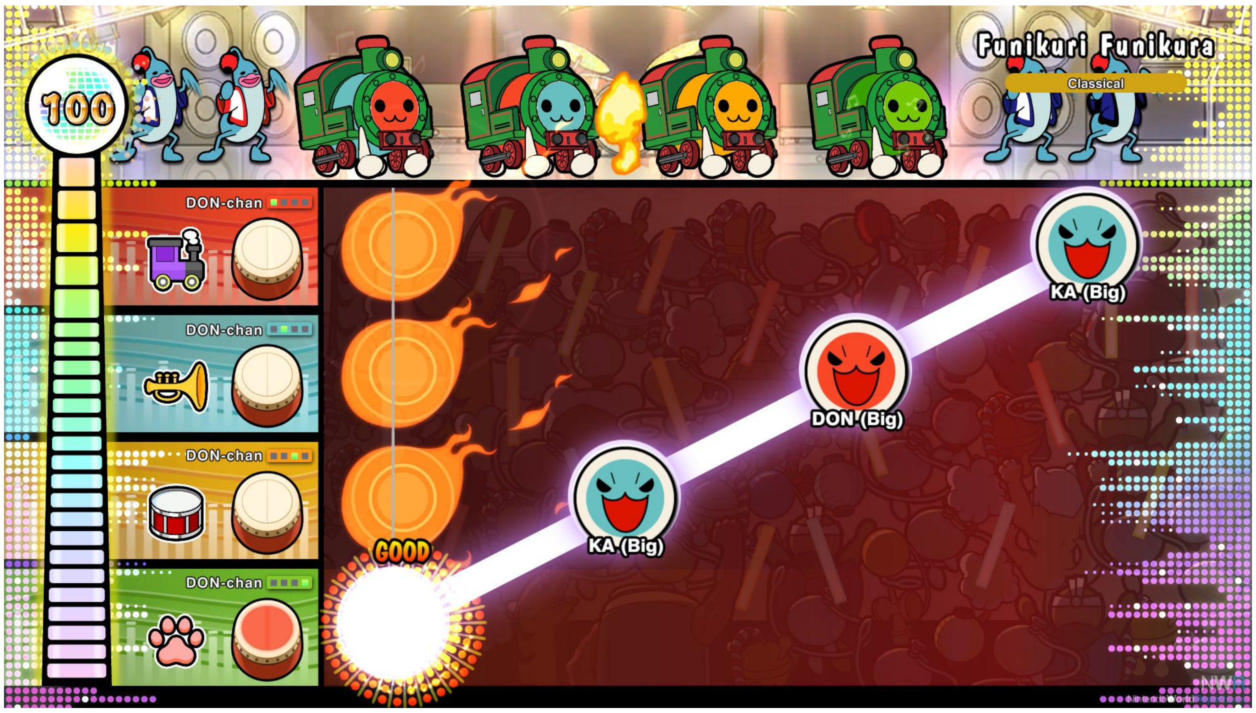
Midterm Project Proposal

Andy Xu, Jiangbo Shen

<u>Basic Idea:</u>
This game is inspired by the musical based games we have played before. You will probably find the model of our idea is very similar to 'Taiko no Tatsujin' or 'MuseDash', but we will make it a little bit different and add some other interesting features.

First of all, the user will be prompted to select a character to start at. The character will be displayed during the actual game, and ideally, it can make various responses according to different conditions in the game. When the game starts, random notes following the background music will be floating around the canvas. The user has to hit certain keys within a certain amount of time in order to receive points. On the other hand, if the user misses any notes, they won't get a score for that particular note. If the user misses a certain amount of notes, they lose the game directly.



Advanced Features:
We want the user to be able to select the number of players in our game. There will be a maximum of two users who can play the game locally at the same time. Their scores will be compared and the one with the highest score will be the winner.

In both ‘Taiko no Tatsujin’ and ‘MuseDash’, the players only use two keys from the keyboard to control the game. We want to make this a little more complex in our game. Instead, the user will be asked to use left, right, up and down keys (or WASD) to control the game. If the player selects solo mode, we might add a feature so that the user can control using their mice.

Instead of making the notes floating around the canvas, the most ideal solution would be to make them coming from one direction. It could be horizontal or vertical direction, based on how we will implement this feature. In addition, there will be a scoring area where the users can only get points when the notes reach the area.

Use an online database or browser’s local storage to create a ranking system where the players are able to check the top 20 scores from the history.

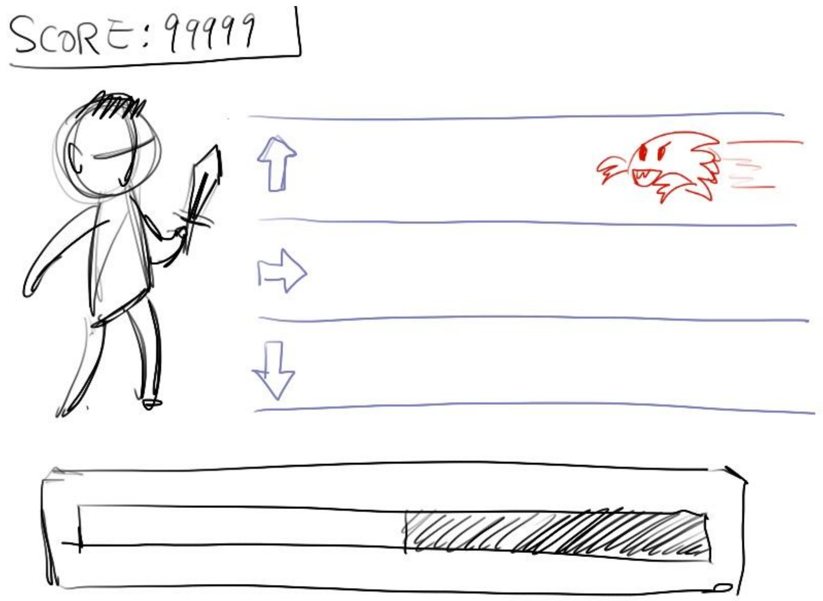
Challenge:
I believe the most challenging and exciting part of this project is how to make the notes follow the beat or rhythm of the background music. To make this happen, we need to set a time when the notes will show up on the canvas. But due to the limitation of our knowledge by far, these times need to be manually measured.

10 /17 Project Update #1

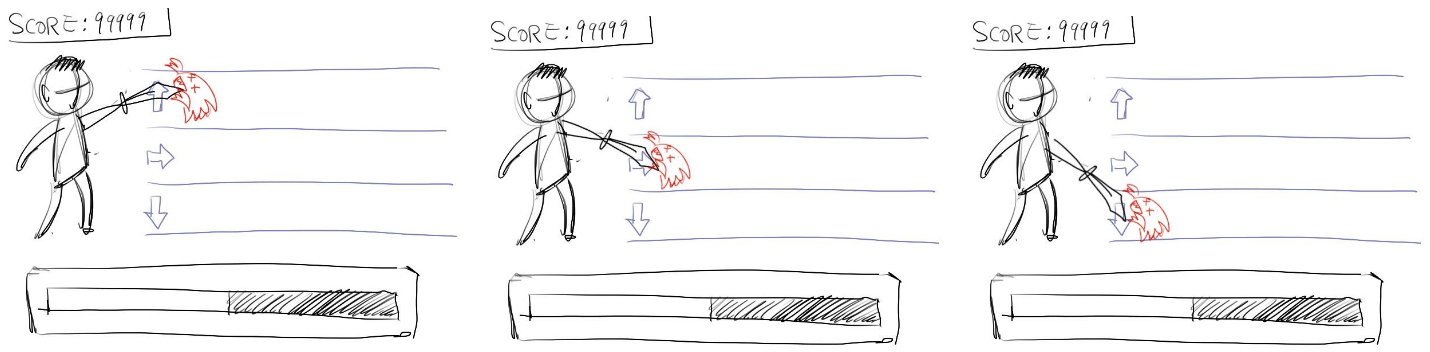
We decided to make this game to have adventurous, monster-fighting visualization while maintaining a rhythmic gameplay. We will create a figure to serve as the game's main character. The character will stay on the left side of the screen, while enemies will appear from the right side and run towards the character. The player gets a point whenever the character successfully hits a monster (that is, from the player’s point of view, press the correct key at the right moment following the music rhythm). When the player misses a hit, the enemy will cause damage to the character’s health bar. The game will end when the player goes through the whole song or when the character’s health turns 0.

Tentatively, we decide to have the enemies move in three rows, and three arrow keys corresponding to each row will trigger regular hits to the enemies. We also plan to make the character own an ability - it will charge when the character makes successful hits consecutively. When the ability is charged, the player can hit the space bar to enter “ultimate mode”, that is, in a limited time period, the character’s attack will hit enemies in all three rows, regardless of which key the player presses.

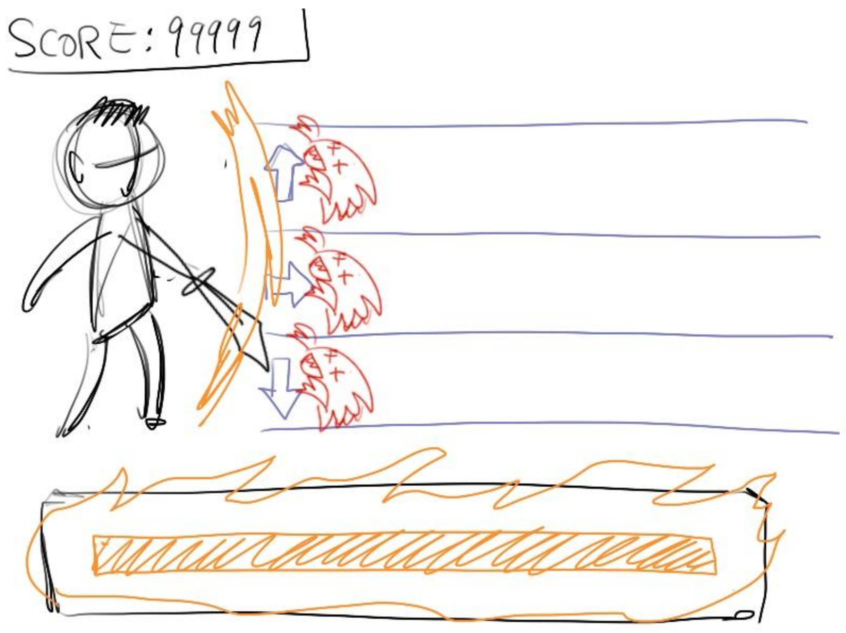
The mechanics of the spawning order of enemies will be random for each game, so the player will have a different experience even if they choose to play the same song repeatedly. The enemies will be stored in an array, and they will be spliced out when they receive hits or when they attack the character.



Game interface setup



Conditions when the character successfully hits an enemy



The ‘Ultimate Mode’ Condition

This is a rough rubric for how we are going to implement the game:

---

1. **Set Up the Environment:**
   - Creating a starting page using javascript and DOM.
   - Include the p5.js library in the project.

2. Create the Canvas:

```
function setup() {
            createCanvas(windowWidth, windowHeight);

}
```

3. Load Music and Background:

```
let song;
let backgroundImg;

function preload() {
            song = loadSound('path/to/the/music.mp3');
            backgroundImg = loadImage('path/to/the/image.png');

}
```

4. Create Drum Elements:

Represent the drums on the screen. We use simple shapes like circles in the early phase.

```
function drawDrum(x, y, diameter) {
            ellipse(x, y, diameter, diameter);
}

// the drum will be replaced with various

// images instead of circles in the end.
```

5. Handle User Input:

Use the ‘mousePressed’ function or ‘keyIsPressed’ function to detect when the user hits on a drum. We will also use the ‘dist( )’ function to check if the click is within a certain range of the drum.

```
function mousePressed() {
            let distance = dist(mouseX, mouseY, drumX, drumY);
            if (distance < drumDiameter / 2) {
                        // User clicked on the drum.
                        // User scored a point.
            }

     }
```

6. Rhythm Logic:

Use the ‘song’ object to play the music and synchronize the game actions with the music beats.

```
function draw() {
            // Any potential game logic likely to be used.

     }
```

7. **Scoring System:**
   Keep track of the player’s score based on their performance in hitting the drums on time.

This will explain in detail how we can synchronize the game actions with the music meats

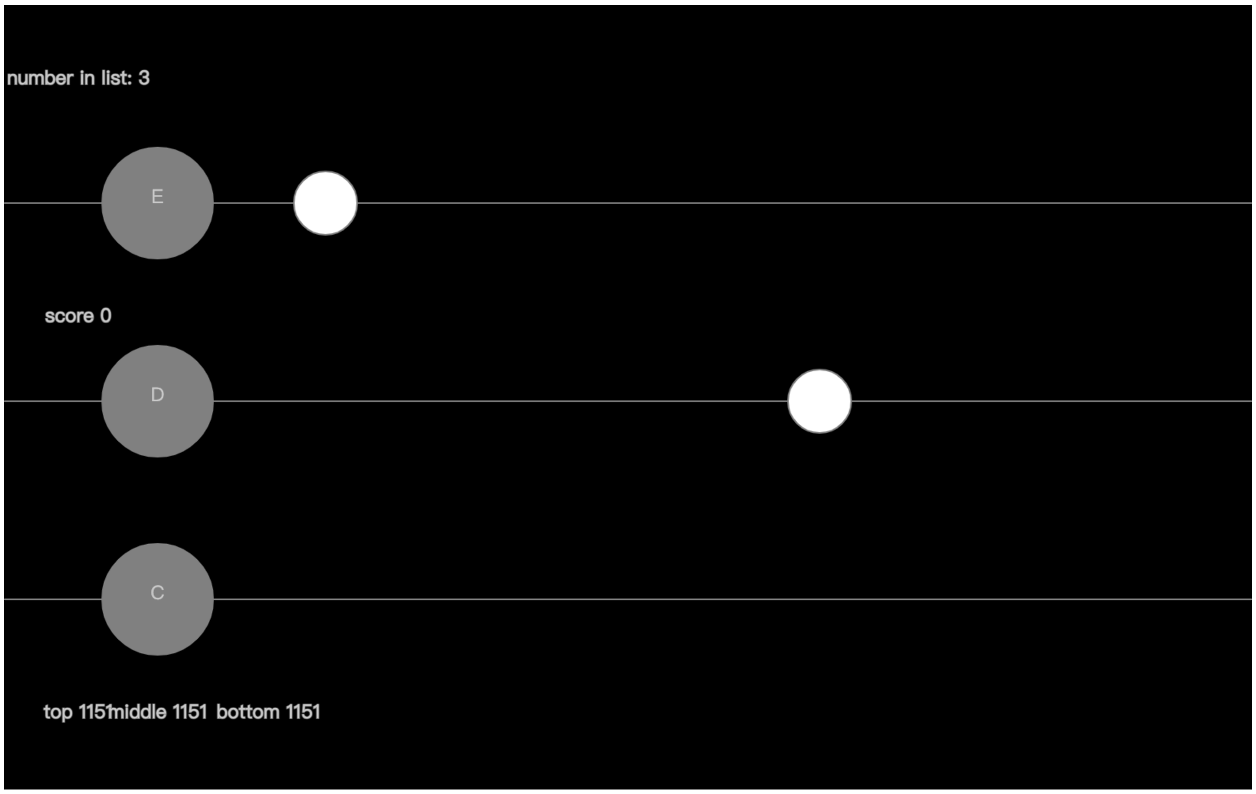
---

Synchronizing game actions with music beats involves tracking the progress of the music and triggering actions (such as hitting a drum) at specific points in the music. According to the p5 documentation, there is a ‘currentTime( )’ method of the sound object to get the current playback time of the audio.

```
let song;
let drumX = 200; // example drum position
let drumY = 200; // example drum position
let drumDiameter = 50;

function preload() {
    song = loadSound('path/to/your/music.mp3');
}

function setup() {
    createCanvas(windowWidth, windowHeight);
}

function draw() {
    // Check if the song is playing
    if (song.isPlaying()) {
        // Get the current playback time of the song
        let currentTime = song.currentTime();

        // Check if the current time matches a beat in the music
        if (isBeat(currentTime)) {
            // Trigger an action, e.g., play a sound, increase score, etc.
            playDrumSound();

            // You can also add visual feedback, like changing the color of the drum
            fill(255, 0, 0); // Set fill color to red
        } else {
            fill(255); // Set fill color to white if there's no beat
        }

        // Draw the drum
        drawDrum(drumX, drumY, drumDiameter);
    }
}

function isBeat(currentTime) {
    // Implement the logic to determine if the current time is a beat
    // This could involve using a pre-defined array of beat timings, for example
    // For simplicity, let's assume a beat every 1 second
    let beatInterval = 1;
    return currentTime % beatInterval < 0.1; // Allow for a small margin of error
}

function playDrumSound() {
    // Add logic to play the sound associated with hitting the drum
    // Use p5.js functions like play(), stop(), etc., to control sound
    // For example:
    // drumSound.play();
}
```

10/20 Update

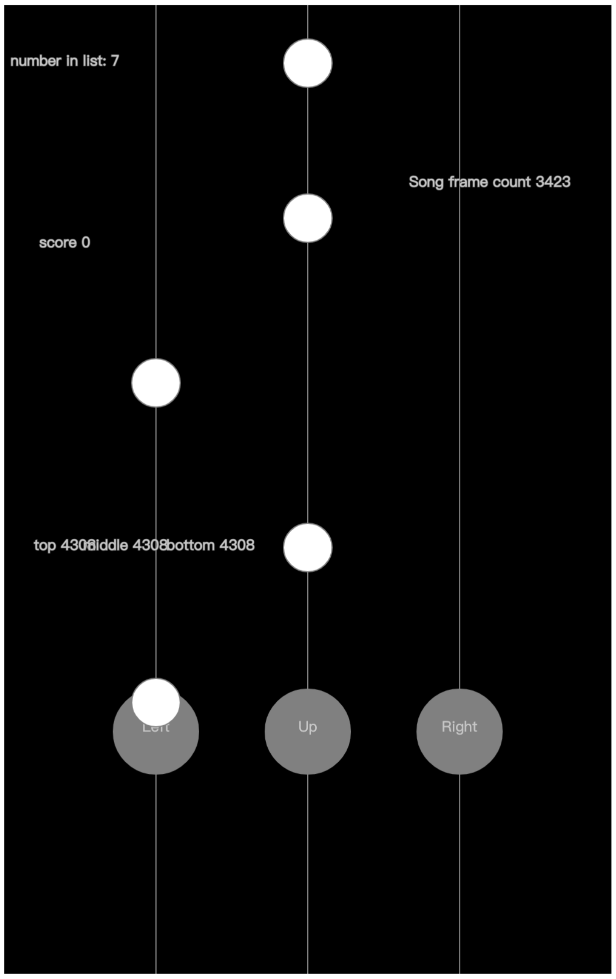
We wrote a brief prototype to test the basic mechanics of a rhythm game with a short piece of music for testing purposes. We generate music notes using the PeakDetect function, which we think is pretty accurate in detecting the beats of the music. However, we still need to manually add additional notes to make the game more interesting, because the PeakDetect function only detects the loudest beats in the music and returns values at a constant rate. We are also planning to create a more dynamic scoring system that could better reflect the accuracy of the player's hit on the notes while playing.


```
number in list: 3
score 0
top 1151middle 1151  bottom 1151
```

<u>10/30 Project Update #2</u>

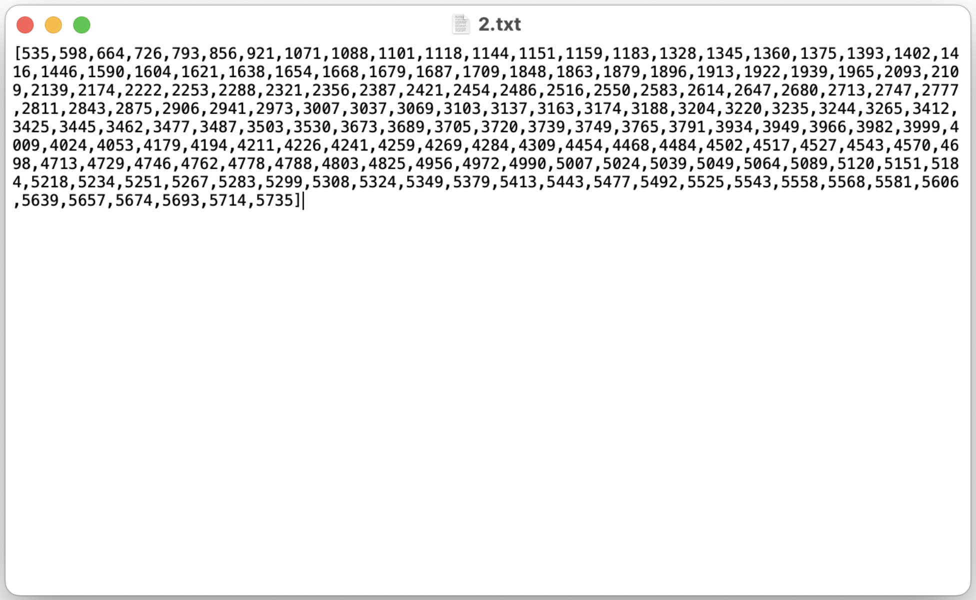
A few changes have been made.

Firstly, we made a vertical alternative of the game that triggers hits with arrow keys at the bottom of the screen, while the notes appear from the top. We feel that the vertical layout version, compared to the previous horizontal version, has a better player experience because it puts the player's hand in a more comfortable gesture on the keyboard.



Secondly, we attempted to improve the note-generating system of our game. We tried both the PeakDetect function and the amplitude function. The sequence of notes generated from these functions is quite accurate in reflecting the actual beat of the music, but in fact, they are not ideal when it comes to playing. When playing rhythm games, people tend to pay more attention to the melody of the music, instead of the rhythm. So the optimal solution might be to manually create the series of notes while utilizing the functions as a reference to adjust the notes to match the music more accurately.

To make the process easier, we wrote a mini-program. This program lets us manually record specific moments where we think the note should appear while the music is playing, and store the frame numbers inside an array. We could hit the space along with the melody and rhythm of the music to record the numbers. When finished, we can download the txt file of the array. The mini program could be accessed inside the “manual_beat_recording” folder.


2.txt

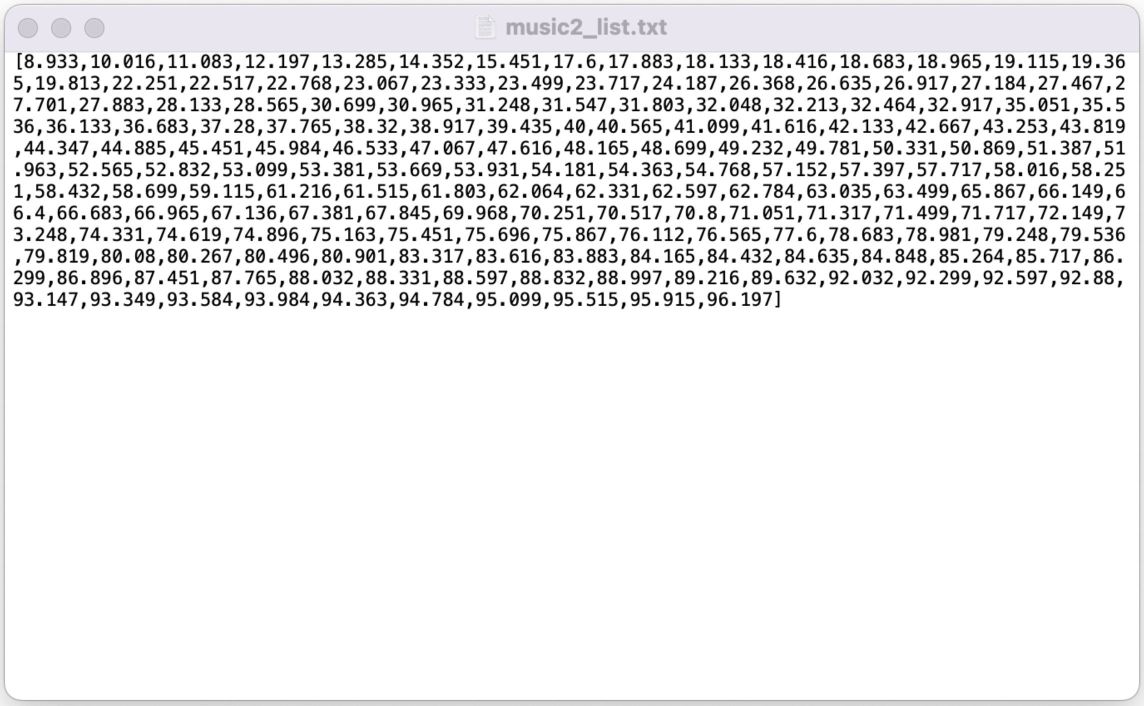
```
[535,598,664,726,793,856,921,1071,1088,1101,1118,1144,1151,1159,1183,1328,1345,1360,1375,1393,1402,14
16,1446,1590,1604,1621,1638,1654,1668,1679,1687,1709,1848,1863,1879,1896,1913,1922,1939,1965,2093,210
9,2139,2174,2222,2253,2288,2321,2356,2387,2421,2454,2486,2516,2550,2583,2614,2647,2680,2713,2747,2777
,2811,2843,2875,2906,2941,2973,3007,3037,3069,3103,3137,3163,3174,3188,3204,3220,3235,3244,3265,3412,
3425,3445,3462,3477,3487,3503,3530,3673,3689,3705,3720,3739,3749,3765,3791,3934,3949,3966,3982,3999,4
009,4024,4053,4179,4194,4211,4226,4241,4259,4269,4284,4309,4454,4468,4484,4502,4517,4527,4543,4570,46
98,4713,4729,4746,4762,4778,4788,4803,4825,4956,4972,4990,5007,5024,5039,5049,5064,5089,5120,5151,518
4,5218,5234,5251,5267,5283,5299,5308,5324,5349,5379,5413,5443,5477,5492,5525,5543,5558,5568,5581,5606
,5639,5657,5674,5693,5714,5735]
```

We copied and pasted the array into our main program. We adjusted the numbers in the array by an offset value to make sure that the notes appeared before their corresponding beat in the music and had time to move into the canvas. We also set up a counter in the program that adds up one by each frame once the music starts. We used a for loop to inspect every number in the array within the draw function, and a note would be produced each time the number in the array matched up with the counter.

However, we found another issue while playing. The notes matched well with the music in the first 20 seconds but started to go off the beat as the music went on. We both tried a couple of times of music recording in the mini-program and used the results to run the main program to see if there were human-measure errors. The problem still exists, although we are convinced that our recordings were pretty accurate on the beats. We guess the reason for this was that while the timing of the music is always consistent, the frame rate of the canvas is not consistent. The frame rate varies more or less around 60 per second. Therefore, the variation in the FPS might create inaccuracy when we use the frame count as the indicator of the timing of notes.

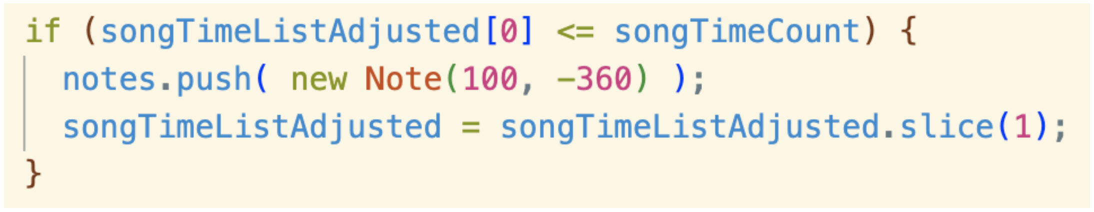
Alternatively, we found another method in p5js called currentTime(), which returns the current position of the music that is playing, in seconds. We hope that this method can be a better measurement of timing than frame counts for the notes and the music. We could use the same mini-program to gather the timepoint information. We will keep working on this in the next day or two to see if the method works.

10/31 Update:
The currentTime() method worked. We recorded the timepoints with the current music time rounded to 3 decimal places, and the notes will appear in our main program as accurate as we did in the recording phase.



```
[8.933,10.016,11.083,12.197,13.285,14.352,15.451,17.6,17.883,18.133,18.416,18.683,18.965,19.115,19.36
5,19.813,22.251,22.517,22.768,23.067,23.333,23.499,23.717,24.187,26.368,26.635,26.917,27.184,27.467,2
7.701,27.883,28.133,28.565,30.699,30.965,31.248,31.547,31.803,32.048,32.213,32.464,32.917,35.051,35.5
36,36.133,36.683,37.28,37.765,38.32,38.917,39.435,40,40.565,41.099,41.616,42.133,42.667,43.253,43.819
,44.347,44.885,45.451,45.984,46.533,47.067,47.616,48.165,48.699,49.232,49.781,50.331,50.869,51.387,51
.963,52.565,52.832,53.099,53.381,53.669,53.931,54.181,54.363,54.768,57.152,57.397,57.717,58.016,58.25
1,58.432,58.699,59.115,61.216,61.515,61.803,62.064,62.331,62.597,62.784,63.035,63.499,65.867,66.149,6
6.4,66.683,66.965,67.136,67.381,67.845,69.968,70.251,70.517,70.8,71.051,71.317,71.499,71.717,72.149,7
3.248,74.331,74.619,74.896,75.163,75.451,75.696,75.867,76.112,76.565,77.6,78.683,78.981,79.248,79.536
,79.819,80.08,80.267,80.496,80.901,83.317,83.616,83.883,84.165,84.432,84.635,84.848,85.264,85.717,86.
299,86.896,87.451,87.765,88.032,88.331,88.597,88.832,88.997,89.216,89.632,92.032,92.299,92.597,92.88,
93.147,93.349,93.584,93.984,94.363,94.784,95.099,95.515,95.915,96.197]
```

We did run into some problems while testing with this data though. In our main program, we were trying to match up the actual music's 3-decimal-places-rounded current time with the ones in the list, but we ended up missing a lot of notes. We assume that the round() function sometimes skips some values, especially when we use the "==" sign to match the numbers. (for example, 8.9335 seconds rounds up to 8.934, which is not equal to 8.933 in the first value of the list in the screenshot). So, instead of using a for loop to inspect every value in the list and match them up with the "==" sign, we used a method that uses the "<=" sign to indicate the moment the current music's time EXCEEDS the first values in the list. If the current time value exceeds the first value in the list, we slice that value out and move every value's position forward by one, and we keep going until the entire list is sliced out with every single note successfully generated.



```
if (songTimeListAdjusted[0] <= songTimeCount) {
  notes.push( new Note(100, -360) );
  songTimeListAdjusted = songTimeListAdjusted.slice(1);
}
```

FINAL DOCUMENTATION

The game is inspired by the musical-based games we have played before. The model of our idea is very similar to 'Taiko no Tatsujin' or 'MuseDash', where players are challenged by their sense of rhythm and hit music notes at certain time points by listening to the background music. The synchronization of music with the notes and the development of user experience through visual elements are crucial to the success of such games.

The foundation we have to build when constructing the game is the mechanics to create notes that correspond to the rhythm of the background music. We did some research on existing p5 functions. For example, the p5.peakDetect() function detects the loudest sounds of music within a given interval of frequency and returns true for every detection. We assumed that these functions would make our note-making process easier because we would not have to manually produce each note to sync with the rhythm of the music.

We wrote a brief prototype to test how the peakDetect() function works with a piece of testing music we chose. The prototype consisted of single geometric shapes. It would generate a circular music note from the left side of the screen and move to the right every time the peakDetect() function returned true when the music was playing. When investigating the music notes that were generated by the function, we realized that the sequence of music notes felt weird. The rhythm of the music that corresponded to the computer-generated notes was different from the rhythm we heard in human ears. Research also showed that most well-known music note games rely on manually creating sequences of notes for a better game experience. Therefore, we had to think of an efficient method to generate notes that are more closely to the human’s sense of rhythm (instead of the computer’s sense).

We chose to trust our ears. We built a mini-program that helped us to manually record time points of the notes. Inside this program, we could record specific time points where we think the music note should be generated while the music is playing by hitting the space key, and in the meantime set up the counter to record the frame number elapsed as the music plays. Each time you hit the space key, the corresponding frame count number is stored in the array. When finished recording, we could download the txt file of the array, and there we could successfully obtain the time point data of each music note.

We recorded the testing music and pasted the complete array into our main program. We set up another counter in the main program that counted the number of frames elapsed since the music started playing and used a for loop within the array to generate music notes whenever a value corresponds to the counter. But although the notes were successfully generated, they went out of sync with the music. We assume that using frame count as the unit for recording is risky and unstable because the frame rate would fluctuate around 60 per second. Therefore, after some research, we found a function called currentTime() that accurately returns the current time of the music that is playing, in milliseconds. We switched the unit for recording from frame count to seconds, and the notes generated were complete and in sync with the music perfectly. It took us weeks but the biggest issue of the game was finally resolved at this point.

The next most important aspect of the game was to create appealing visuals. The player’s experience with rhythm games relies heavily on graphic presentation because they will need intensive and frequent positive/negative feedback while hitting the notes when playing. For example, many music-based games use bright yellow or green to indicate an accurate hit of notes and less saturated colors to indicate an inaccurate hit or a miss from the player. Plus, they often use “PERFECT”, “GREAT” or “GOOD” to indicate the accuracy of note-hitting. We absorbed our competitors’ features and divided the grading of hit accuracy into three levels: “PERFECT” with a bright yellow color, “GOOD” with a bright green color, and “OKAY” with a gray color. A higher accuracy level provided a higher score to be aggregated over the play. In addition, we built a particle system that would create a splashing effect upon hitting notes, which provides a better visual impact and more positive feedback to the players, to improve the game experience.

Finally, we put the core of our game into a storyline that differentiates from most rhythm games: we combine a traditional music note game with an RPG setup. The player has three adventures to choose from, and they each have a different background setup and music to play from. During the game, there will be monsters that dash toward the player-controlled knight in three columns, and the knight should hit the monsters with the sword. To successfully hit a monster, you will have to press one of the three keys (J, K, or L) at the right point in time (that is, hitting the “notes” along with the rhythm of the music, in the player’s perspective).

We created the graphics ourselves because we believe that it’s better this way considering the unity in style and color of different graphic elements. Plus we have a team member majoring in art so this process should be quite efficient. We used a software called Procreate - an iPad software that utilizes Apple Pencil to create digital artworks. And here we go! Here comes the launch of our game!

In future development, we would like to add more features such as the character's abilities or even more songs to let the players choose from. We took too much time figuring out the mechanics of the note-generating system so we believe that we could make the game even more fun if we got more time. We would also like to receive feedback from players after it is launched to see if they think the notes are out of sync with the music or any other suggestions that might help us improve the game.