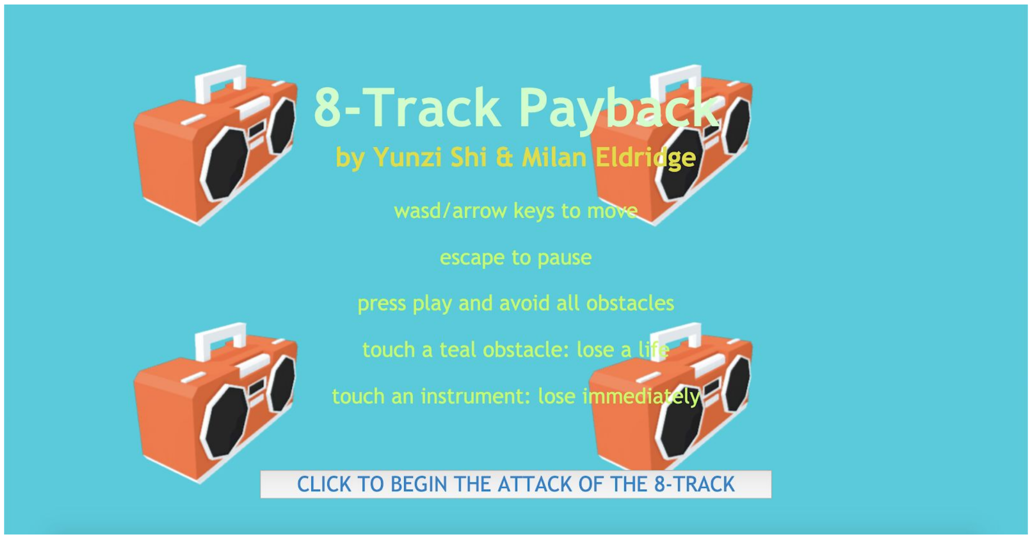
8-Track Payback

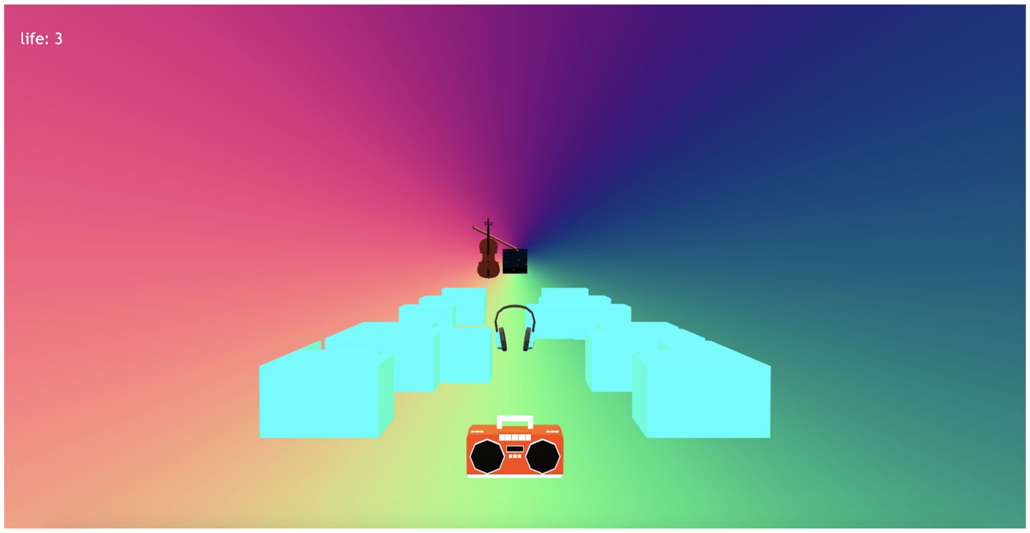
COS426 Final Project
Milan Eldridge, Yunzi Shi
May 12, 2020

Abstract

8-Track Payback allows the player to visualize and interact with music they are hearing. Upon entering the game, the player can select the music to be played from a list. The player then controls an avatar through a tunnel-like scene, trying to dodge the 3D obstacles coming from all directions that are generated from the music being played. Getting hit by an obstacle costs the player's lives. The player loses the game when all lives are lost, and wins when they survive a complete piece of music.



8-Track Payback
by Yunzi Shi & Milan Eldridge
wasd/arrow keys to move
escape to pause
press play and avoid all obstacles
touch a teal obstacle: lose a life
touch an instrument: lose immediately
CLICK TO BEGIN THE ATTACK OF THE 8-TRACK



Introduction

Goal

The goal was to create an interactive game that visualizes and initiates fun interactions with music that by nature does not have any visible form. While many games feature players interacting with visual components, we wanted to create a game that incorporated an aural component that influenced the visual characteristics of the game. This game is designed both for those who are passionate about music and interacting with its digital format and for those who would like to understand more about music from visualizing its properties. Furthermore, this game is for people who enjoy a fun challenge and have a thirst for winning.

Previous Work

Numerous music games exist in the forms of arcade games, video games, or mobile games. On the one hand, Rhythm games like *Taiko no Tatsujin* lets the player replicate the beats for the music. Games such as *Tap Tap Revolution*, *Guitar Hero*, and *Rock Band* also allow the player to follow along with the rhythm of the music by 'playing' the notes of the song. *Dance Dance Revolution* designs steps according to the music so that the player gets to dance according to the music. *Just Dance* takes this interaction a step further and allows the player to involve more of their body in the game play and to interact with the song being played in real 3D space. Most of the works, however, only extrapolate from the original music or only let the viewer interact with some components of music, such as the rhythm.

On the other hand, music visualization including waveforms and frequency bar graphs represent more characters of music. While generated live as the music is played, they do not allow user interaction. For example, many music artists post audio visualizers to accompany their music in-lieu of a music video. Often dance venues and concerts feature moving lights that emulate the musical characteristics of the song being played.

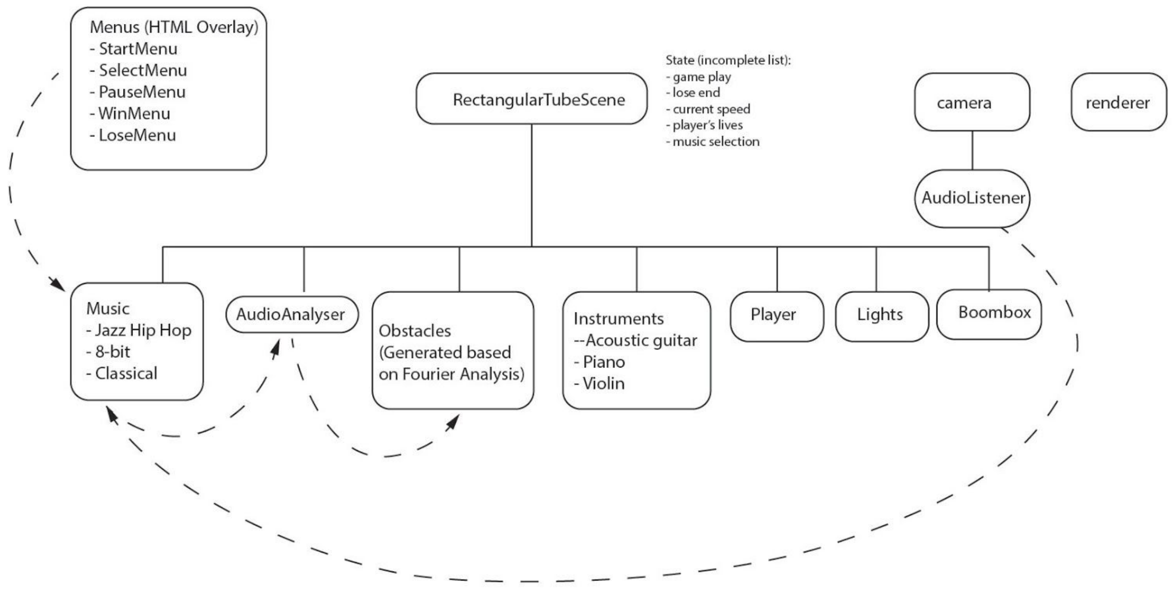
Recently, there have been many other data visualizations and sound sequences created using various characteristics of music in sound. For example, Jordan Wirfs-Brock, an information science Ph.D. student at the University of Boulder Colorado used sonification in order to create a new way to "graph" the stock market. The sequence represents the state of the Dow Jones Industrial Average by using pitches in order to correspond to volatility and the final closing price. There are many other examples of sonification from a wide variety of fields in which data is converted into music in order to convey information without the use of speech-oriented audio or visual components. Audio visualizations such as these exemplify the possible bridge that music can act as when used in conjunction with other, primarily visual, fields such as data representation.

Approach and Methodology

To achieve to goal of having the user interact with the music, we have implemented:

- The obstacles (audio visualizer and fast Fourier Transform, movement, collision detection)
- Scene setup
- Avatar that moves according to user's input
- Gameplay (

Below is a diagram showing the overall architecture of our game:



Obstacles

In order to visualize the audio we used the audio analyzer provided in Three.js. This analyzer was able to take a piece of music and Fast Fourier Transform size in order to create the analyzer for the music. We then used the analysis data in order to create obstacles that the player would need to avoid in order to have a successful run through the game.

Each time the scene gets updated, the frequency data is passed to each of the obstacle objects created. This data is translated to a factor for interpolating between 0-255 of frequency. With the factor, the obstacles are morphed according to the frequency data using Tween. With each update, the obstacles also move towards the player in the negative z direction in succession. When an obstacle becomes too close to the camera/screen, it is moved back along the z direction.

Each obstacle has a function that determines whether the obstacle is in collision with the player based on whether the bounding boxes intersect. If a collision is detected, it costs the player one life. The obstacle is moved further back along the z direction (to avoid multiple collisions with one obstacle). If the life of the player drops to 0, the player loses the game.

In addition to the obstacles created from the music, we also created a secondary type of obstacle in the form of music instruments that moved towards the player in the perpendicular direction. Thus, the player must avoid the audio analysis bars in addition to the instruments that are approaching. The movement forward is implemented with a Tween. However, we do not feel that there are too many objects in the scene either and we have thus found a 'Goldilock's Zone' for the object quantity in our scene. As a result of this, while playing the game, the player is also able to enjoy the audio visualization. Therefore, we feel that the game is able to be both challenging and engaging for the player.

As the audio analysis generated obstacles are relatively difficult to predict, we added a 'lives' feature that gave the player three lives in relation to the audio generated obstacles. The aforementioned approach describes the main crux of the gameplay.

Scene

Many classic music video games have a tunnel-like scene in which the objects that the player needs to interact with come in succession. This format corresponds to the properties of music: being able to visualize the objects to come, the player can pick up the beats and patterns of a song or music more easily. We adopted a similar approach in this game. In RectangularTube.js, we created a tube with geometry from THREE.js, which becomes the major feature of our scene. We used MeshNormalMaterial for creating the tube mesh to give the tunnel a fancy color. The player's movement is confined to this tube where the obstacles appear. We also added a Boombox, whose gltf model was also obtained from https://poly.google.com/, in order to act as a reference point for the location of the avatar and the obstacles approaching the avatar.

Avatar

We used a gltf model of a pair of headphones published by 'Poly by Google' on https://poly.google.com/as the avatar of the player. The avatar can be controlled by keyboard since restricts the player's movement to limited directions, making it easier for the game to determine and respond to the player's actions.

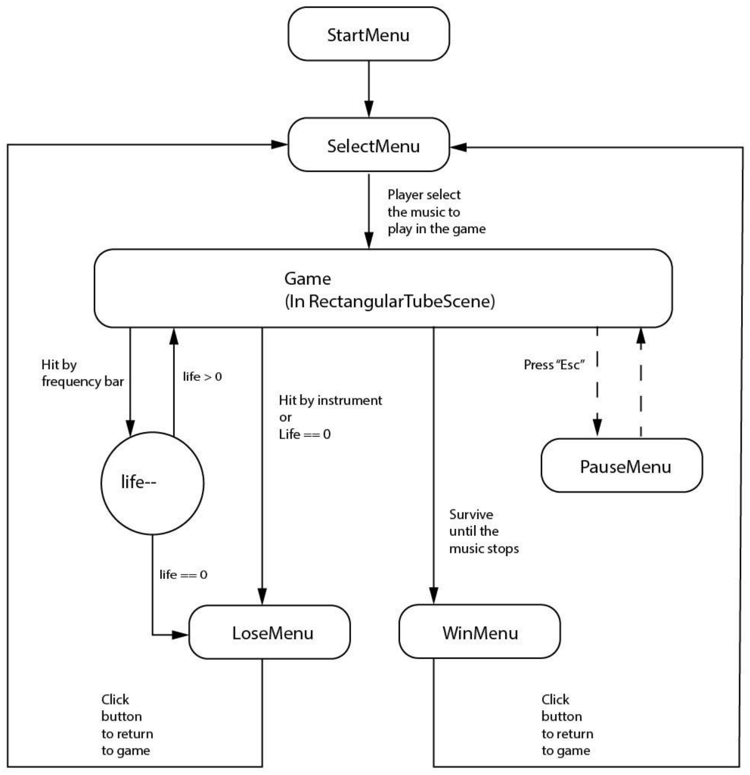
Gameplay

The game uses a series of informative menus to guide the player. All the menus are created via HTML overlay. We used a CSS stylesheet in order to create styles for the classes found within the various menus.

By clicking the button on the start menu, the player gets to choose the piece of music to play with from the select meu. By clicking on the respective button, a field in the scene's state called "musicSelect" is set, and the corresponding music is loaded and starts playing automatically.

The game tracks the state of the player and whether the player is losing with fields in the state of the scene. Whenever a collision is detected with an obstacle, the life number of the player gets deducted. When the player loses, the lose menu is overlayed on top of the scene which becomes translucent. It lets the player click to restart the game.
When the selected song ends, the player is congratulated by a win menu, which gives them the option to restart the games again.

In addition to the menus, the scene keeps track of the life, the speed, the game state in its state, and responds accordingly. A typical gameplay flow is shown in the diagram below:


StartMenu
SelectMenu
Player select the music to play in the game
Game (In RectangularTubeScene)
Hit by frequency bar
life > 0
Hit by instrument or Life == 0
Press "Esc"
PauseMenu
life--
Survive until the music stops
life == 0
LoseMenu
WinMenu
Click button to return to game
Click button to return to game

Results and Discussion

We encountered many challenges implementing this game. The first is animating the obstacles. Initially we were removing and recreating new geometries according to the new frequency, this it was computationally expensive. This was resolved by switching to using morphAttributes of the objects and Tween to animate the changing geometries.

Another challenge we encountered was using Tween.js. For example, with Tween, the Tween object continued to run despite the pausing of the render loop. Unfortunately, there was no previously existing pause and play function for the Tween object. As a result of this, we learned that it was important to understand how this object was structured in order to properly use them how we desired.

We also heavily used Three.js. Playing with a 3d scene is interesting but it also introduces problems dealing with perspective. By switching the position of the camera and the player, we tried to allow the player's perspective to be slightly above the avatar so they can see the frequency bars coming.

One of the first things we learned by doing this project was how to use Github in order to work with collaborators. While this presented a challenge at first, once we started to understand how the branches affected my working space, we found it to be relatively simple to work with each other in order to create our game. We also learned more about HTML and CSS. While many of the writeups for the class used HTML, we did not fully understand each of the components. When starting out, it was very useful to be able to use the starter code as a starting off point in addition to the code from some other projects that used Three.js. We did not realize that there existed so many resources online for coders to use as a starting off point when creating a program or game. This was definitely something useful that we realized because after having many of the assignments begin with starter code, we were worried that we would never be able to create a program on my own. Thus, it is comforting to know that there are many resources and libraries online that can serve as a starting off point. We also learned more about modularizing code and certain strategies in relation to separating code into different files. For example, we added a 'state' component to our scene to be able to determine what actions should be taken and which menus should appear if certain obstacles, keys, or buttons are pressed. We also learned more about using models and how to convert .bin files into a format that could be included in the .gltf file as well as how to resize and reposition models from the original.

Conclusion

We reached our goal in general to visualize music and enable interesting interactions with it. More specifically, we were able to successfully create obstacles from the visualization of the music that the user selects and hears throughout the play of the game.

The next step would be to further develop the levels. For example, there could be an increase in difficulty based on the length and analysis of the songs. The levels could begin with a selection of shorter songs and progress to a selection of longer and more complicated songs that would create obstacles that are more difficult to avoid. This would either require adding more songs to be able to be loaded by the game or could potentially be obtained by having the level determine the duration of the music that is played.

Contributions

Milan implemented the rectangular tube, the instruments, the player, and the menus.
Yunzi implemented the camListener, the analysis of audio, the obstacles that correspond to frequency, the music selection menu, and the life counts.
We collaborated on building the rectangularTubeScene.

Works Cited

- Collideoscope https://github.com/ewilden/collideoscope/
  - Many of our initial concepts and approaches were inspired by this project. A huge thank you to the co-creaters!
- Models Imported from - Poly.Google
  - Boombox: https://poly.google.com/view/56XYDxnVVM3
  - Headphones: https://poly.google.com/view/frvTEfwm9Yg
  - Piano: https://poly.google.com/view/7U-93vxPOER
  - Acoustic Guitar: https://poly.google.com/view/afr6GCpce_I
  - Violin: https://poly.google.com/view/4yO-XrJiBlc
- Web Audio API
- Tween API
- Music Excerpts
  - DJ Okawari - You Gotta Be https://youtu.be/h_rM6e7eVh0
  - Kirby's Return to Dream Land Title Theme 8 Bit Remix https://youtu.be/oMgQJEcVToY
  - Offenbach https://youtu.be/9WpncKShGmU