BeatSabers

CSC 631 Multiplayer Game Development

Professor Ilmi Yoon

Spring 2018

Team:

Hung Do

Tatem Hedgepeth

Anirudh Mohan

Carmen Cheung

Justice Chase

Saengduean Calderez

Taylor Marquez

Audio Team:

Jairus Cambe
James Jordan
Sean Thompson
Jake

Revision History

Revision Number	Name	Date
1.0	Design Documentation	2/29/2018
1.1	Audio Added	3/1/2018
1.2	Revised Documentation	3/12/2018
1.3	Final Documentation	5/20/2018

Table Contents

Executive Summary

BeatSabers is an action oriented rhythm-based game played using Virtual Reality (VR) headsets that includes movement of the player’s whole body. We are teaming up with University of California San Francisco’s (UCSF) Neuroscape department, to design a game that makes players more rhythmic. Neuroscape is a neuroscience center at UCSF which combines technology with scientific research to better improve brain function for all. Their hypothesis is that rhythms will make the player better at anticipating which links to cognitive improvements. Physical exercise is the key component to this virtual reality experience. With the combination of both physical aspect and cognitive training the results would yield better improvements than just cognitive training alone. With our adaptive algorithms our game’s difficulty will adapt to the players level constantly.

The player will swing VR-simulated “lightsabers” to slice beats in sync with the rhythm of music with varying levels of difficulty. The game results is primarily going to be used by scientists to check certain parts of improved brain functionality. Our game will simulate a fun and replayable environment. The movement will be simple; swaying arms, moving feet, and maneuvering your upper body. Although there are no intense moves, this will give you a decent workout with a enjoyable time playing the game. The heart monitors will track your heart rate while you are playing the game, the results will determine if the player has reached their potential.Our team with the help of UCSF is well-rounded and excited to be part of this virtual reality game. We are well determined and motivated to make this game happen.

Concept

BeatSabers is structured in a way where a player stands near a translucent wall that has some depth and objects fly towards the player at a constant speed. It is the player’s objective to use his dual lightsaber swords to break these objects within the translucent wall. Points will be given based on precision of breaking the object exactly when the whole object is inside the translucent wall. To provide more flare, there will be robots that occasionally shows up and shoot laser. When the robot appears, all the objects coming at the player shall disappear. At this point, it is up to the player to dodge the incoming laser. Once you dodge or get hit, the robot goes away.

Since it is also a rhythm game, there will be a tune playing in the background and while you are breaking the objects, there will be beats that are properly incorporated with the tune.

Therefore, as you break the objects, the tune will be in perfect sync. This includes the appearance of the robot as it will not be out place and intuned with the music.

We initially wanted the game will be set on adaptivity, as in, as you improve, the game gets harder and as you miss, it get easier. But since we did not have time to get to that part, we have set out game to three difficulty variations of easy, medium and hard that act as variants to the one tune being played in background. The player can choose his difficulty level in the main menu of the game. We also have two soundtracks that players can choose in the menu. The difficulty adaptivity of the music shall be by the increase and decrease of the speed at which the orbs move. If the player hits more than 80% of orbs, the orbs will move faster. If the hit less than 60% the orbs will move slightly slower.

BeatSabers is a time based game and so you will have a certain amount of time to get through the game with the best of your ability. In the end, there will be statistics and analyses on your perfect hit, early hit, miss, lasered, robots destroyed. There will also be specific the hit rate and miss rate based on the length of time of the three difficulty variations that you had throughout the game stored in the database. For Example, in the future, the game will display the total amount of time you played in easy, medium and hard level and calculates the misses and hits rate per beat you had throughout the period of the game.

Specifications

Functional Specs

1. The scientist shall login for the player in order to play the game.
2. Player shall be able to move freely in a designated area.
3. Player shall be able to interact with targets in a ‘reaction area’.
4. Player shall interact with the target by touching them with their sword.
5. Player shall receive the scoreboard at the end of the gameplay.
6. Player shall receive varying amounts of points depending on how accurately they touch a beat with regard to the boundaries of the reaction area.
7. Scoreboard shall consists of score, hit, miss, and total score.
8. Target shall disappear when touching by the player’s sword.
9. Target shall come toward the player randomly.
10. Robot shall be able to shoot the laser.
11. Player shall be able to dodge the laser shooting by the robot.
12. There shall be sound when the sword touch the target.
13. The game difficulty shall be adapts to the player’s performance during the game.

14. The game shall be a time based game.

Non- Functional Specs

1. The game shall use HTC Vive headsets with handle controllers and movement trackers
2. The game shall run on PC platform
3. The game shall provide submersive experience in accuracy, sound, movement and fun

Audio

OVERVIEW

The audio components of BeatSabers comprise two groups, sound effects and music. Sound effects include sounds which correlate with player actions such hovering over menu buttons, swinging the lightsaber, and hitting or missing a target. The music tracks serve as a cue to the player for when to react to incoming notes and when to strike.

IMPLEMENTATION

Sound effects were produced using FMOD and Ableton Live while the music tracks were produced using Ableton Live, FL Studio and AudioHelm.

To incorporate sounds within the game, and audio manager script was created (further information in coding architecture) which imports the required audio files in to the project, assigns them key values and allows any other script in the project to invoke the playing of a sound based on the correlating key value.

AudioHelm was used to facilitate the spawning of note objects based on a midi file which is uploaded to the project. For the music tracks produced by the audio team, the midi file used is a specific track of the song (percussion) which provides note spawning data based on rhythm of the song. For 3rd party tracks and audio files, a custom midi file can be created to provide data for the spawning of notes.

Music Deliverables:

MUSIC

- Menu Theme
- Meltdown (Jungle Theme Song)

<u>SFX</u>

- Menu Sounds
- Lightsaber Sounds
- Flying Robot Sounds
- Target
- Flying Robot Laser Sounds

Use Cases

Casual Player

Bob is a casual player and is looking for some quick entertainment. He launches Beatsabers using SteamVR. Immediately, he is prompted to register an account. He registers an account, then logs on. On the main menu, there consists of start, quit, and unlockables. He chooses the play option to start the game. He plays through the game casually. After the game ends, Bob sees his stats such as how many beats he has hit. However, he can care less about it since he is a casual player. Then, he starts another game for more entertainment by pressing the play again button.

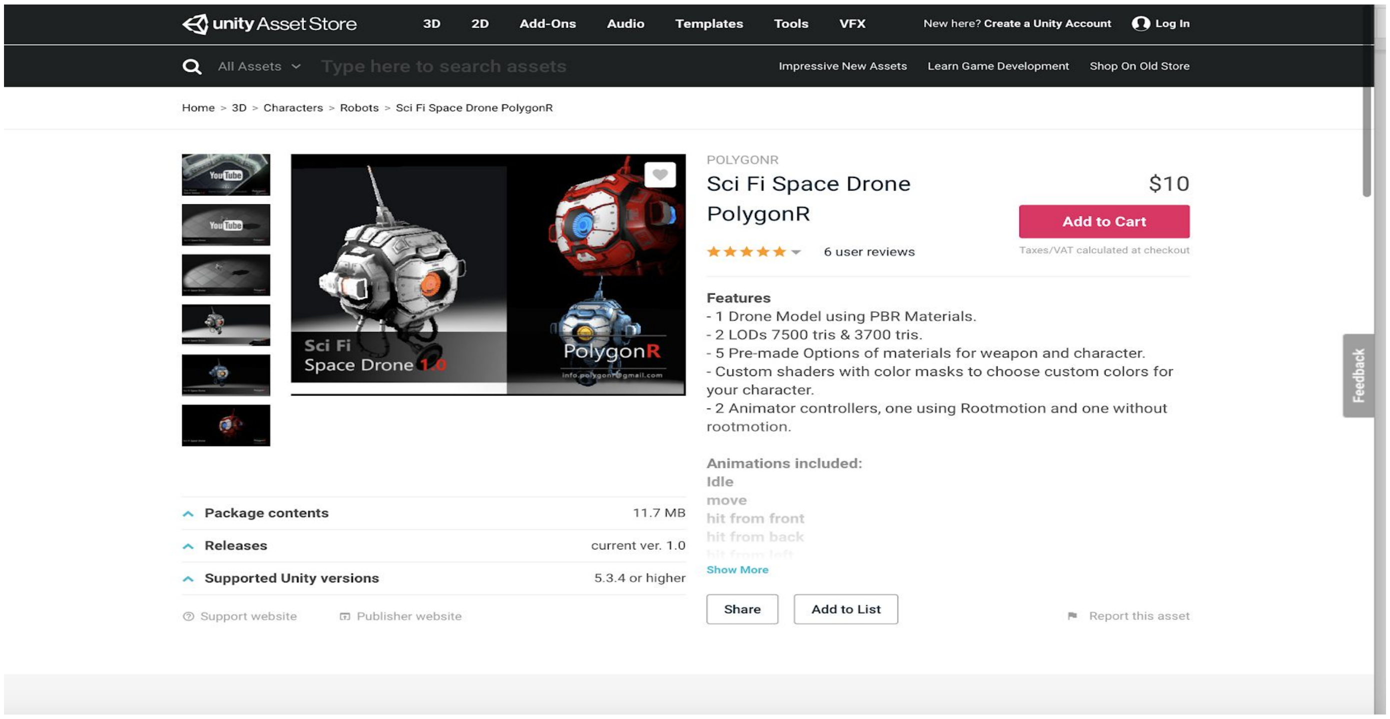
Overachiever

Sandra is known for striving for all unlockables at any game. She launches Beatsabers using SteamVR and is prompted to register for an account. She registers an account and logs in. On the main menu, there consists of start, quit, and unlockables. The first option she clicks on is the unlockables button. She looks at what she can unlock at the moment. Then she starts the game to try and achieve that unlockable. After the game ends, it will prompt that she unlocked a new lightsaber. She goes back to the main menu and checks the unlockable. Oh look, she received a new lightsaber color.

Exerciser

Jordan loves playing games that involve movement and exercise. He launches Beatsabers and is prompted to register an account. He registers an account, then logs on. On the main menu, there consists of start, quit, and unlockables. Immediately, he chooses the play option to start the game. Jordan strives to heat every single beat in order to ramp up the difficulty of the game. He also wants to survive as long as possible. By the end of the game, he is sweating and happy that he is able to achieve a workout from it. He feels that he wants another exercise, so he chooses the play again option.

Models purchased from the Unity Asset Store:



unity Asset Store
3D 2D Add-Ons Audio Templates Tools VFX
New here? Create a Unity Account Log In
All Assets Type here to search assets
Impressive New Assets Learn Game Development Shop On Old Store
Sci Fi Space Drone 1.0
PolygonR
Feedback

Home > 3D > Characters > Robots > Sci Fi Space Drone PolygonR

POLYGONR

Sci Fi Space Drone PolygonR $10

Add to Cart

6 user reviews

Taxes/VAT calculated at checkout

Features
- 1 Drone Model using PBR Materials.
- 2 LODs 7500 tris & 3700 tris.
- 5 Pre-made Options of materials for weapon and character.
- Custom shaders with color masks to choose custom colors for your character.
- 2 Animator controllers, one using Rootmotion and one without rootmotion.

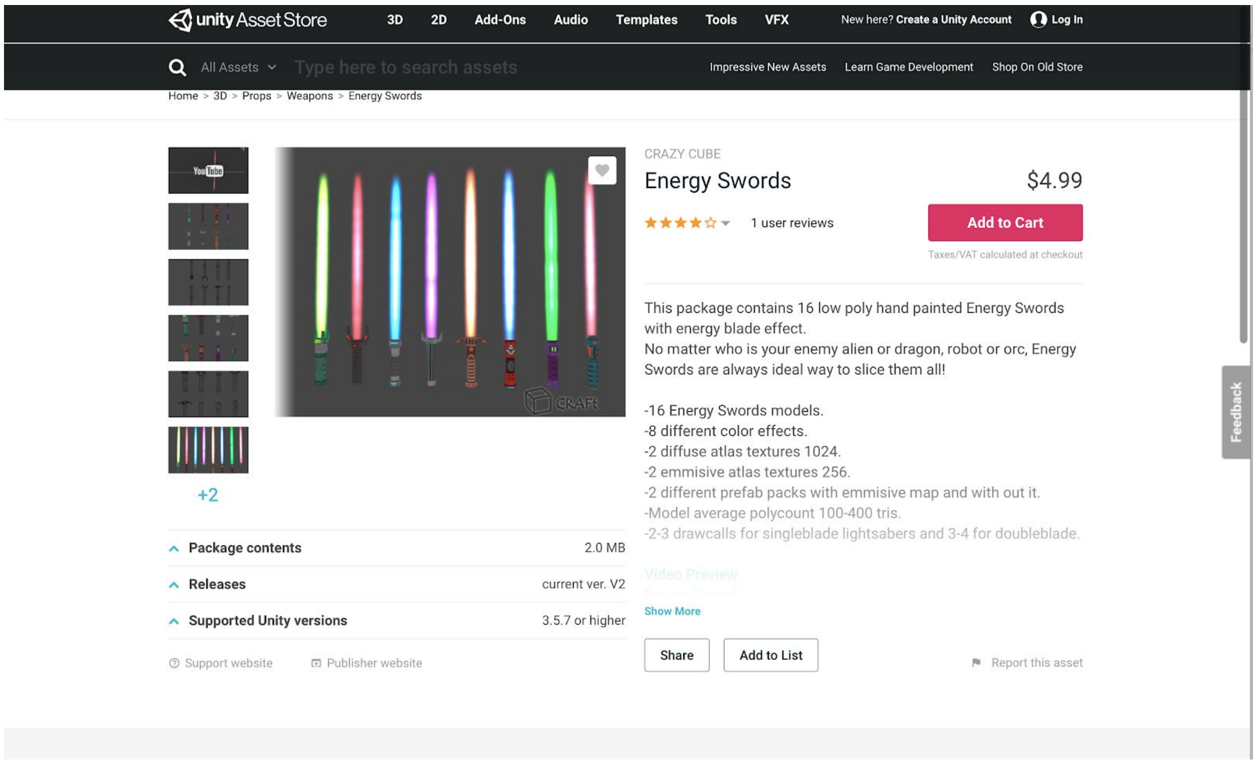
Animations included:
Idle
move
hit from front
hit from back

Show More

Package contents	11.7 MB
Releases	current ver. 1.0
Supported Unity versions	5.3.4 or higher

Support website Publisher website

Share Add to List Report this asset



unity Asset Store
3D 2D Add-Ons Audio Templates Tools VFX
New here? Create a Unity Account Log In
All Assets Type here to search assets
Impressive New Assets Learn Game Development Shop On Old Store
+2
Feedback

Home > 3D > Props > Weapons > Energy Swords

CRAZY CUBE

Energy Swords $4.99

1 user reviews

Add to Cart

Taxes/VAT calculated at checkout

This package contains 16 low poly hand painted Energy Swords with energy blade effect.
No matter who is your enemy alien or dragon, robot or orc, Energy Swords are always ideal way to slice them all!

-16 Energy Swords models.
-8 different color effects.
-2 diffuse atlas textures 1024.
-2 emmisive atlas textures 256.
-2 different prefab packs with emmisive map and with out it.
-Model average polycount 100-400 tris.
-2-3 drawcalls for singleblade lightsabers and 3-4 for doubleblade.

Video Preview

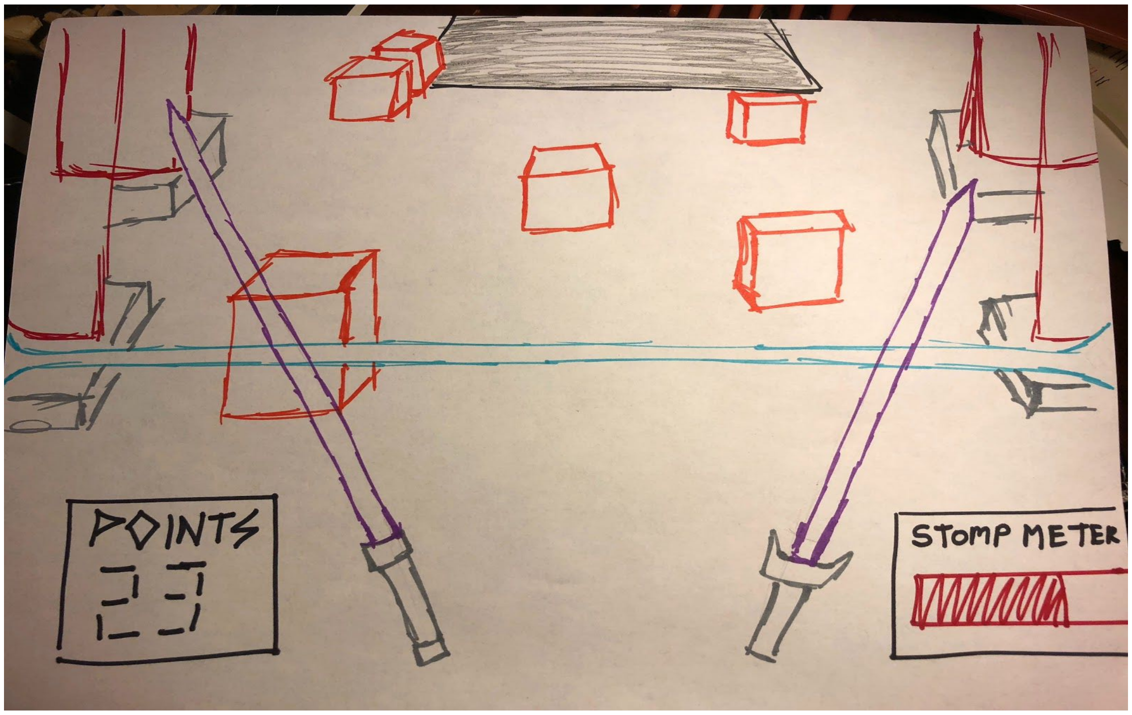
Show More

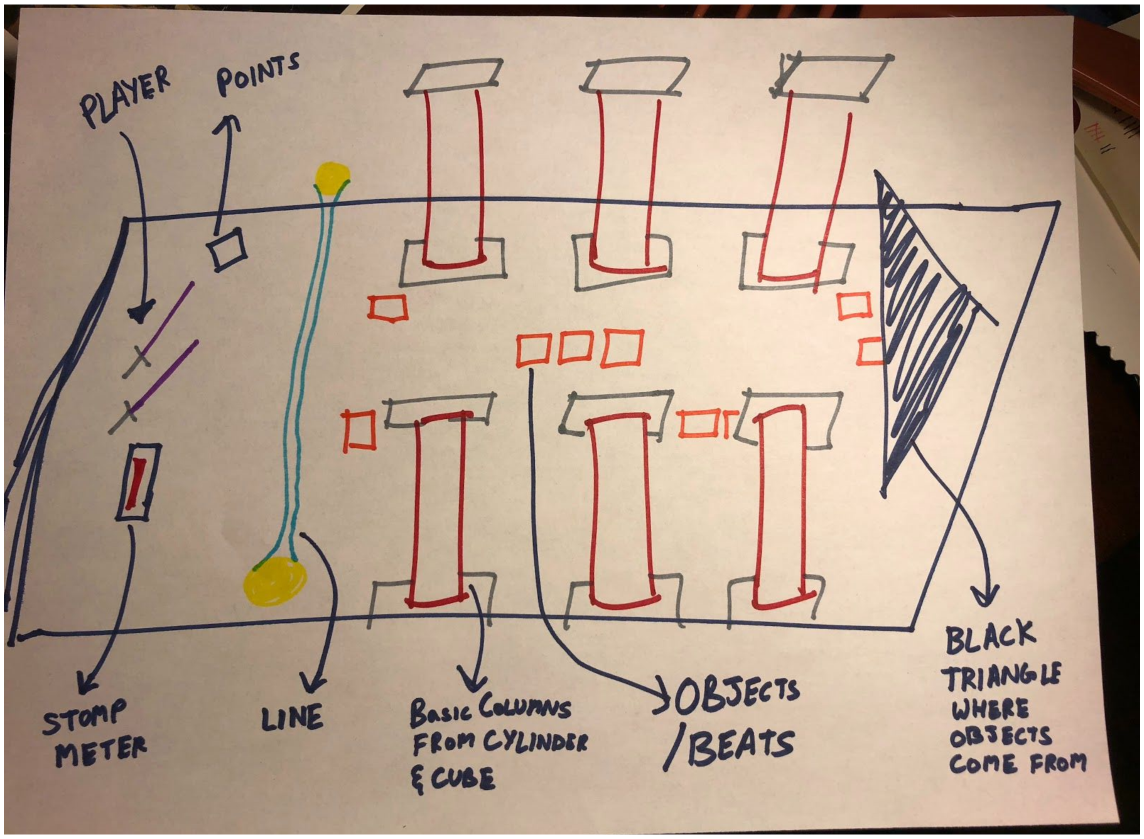
Package contents	2.0 MB
Releases	current ver. V2
Supported Unity versions	3.5.7 or higher

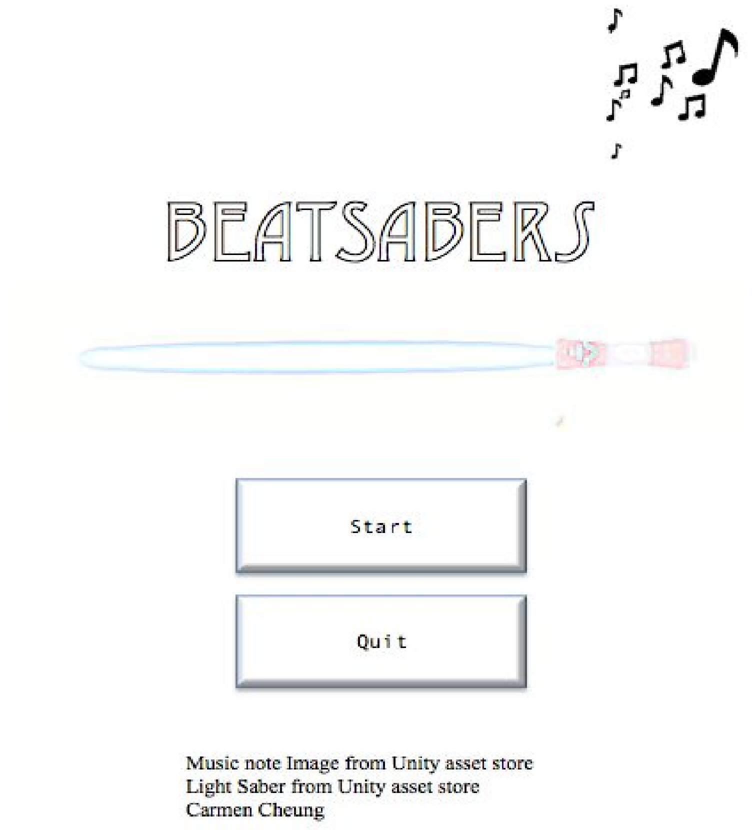
Support website Publisher website

Share Add to List Report this asset

MockUps


POINTS
23
STOMP METER


PLAYER
POINTS
STOMP METER
LINE
Basic Columns From Cylinder & Cube
OBJECTS /BEATS
BLACK TRIANGLE WHERE OBJECTS COME FROM


BEATSABERS
Start
Quit
Music note Image from Unity asset store
Light Saber from Unity asset store
Carmen Cheung

Team Requirements

Priority 1:

1. Implement VR functionality for HTC Vive and controllers to be motion tracked in real time - Justice, Hung
2. Create the game scene and beat object spawn area - Ani, Carmen
3. Create the reaction area and implement accuracy-tracking integration with beat objects - Tatem, Justice
4. Create at least one song for the game - Audio Team
5. Create lightsabers that interact with beat objects - Hung
6. Create robots that shoot lasers - Taylor
7. Implement scoreboard to track player stats - Hung

8. Implement database for player information - Saengduean
9. Create game stages - Ani
10. Implement game menu - Carmen
11. Produce audio assets for menu interaction, lightsaber movement, flying robot actions, beat object interaction and accuracy feedback - Audio Team

Priority 2:

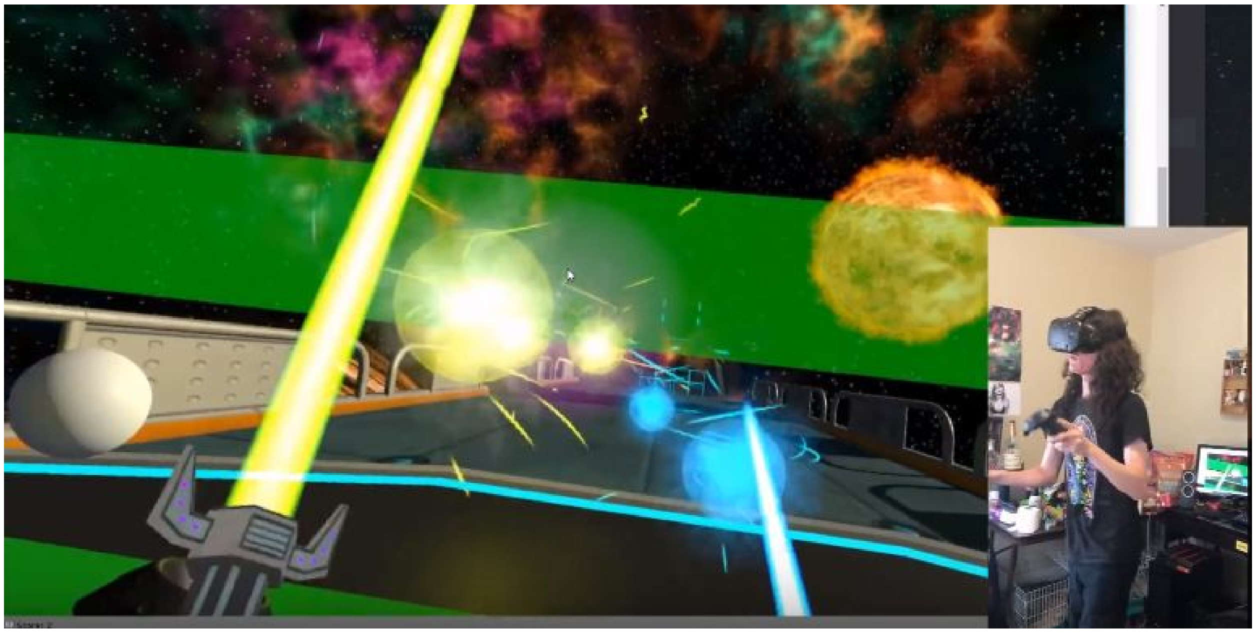
1. Achievements
2. Unlockable items22
3. Leaderboard

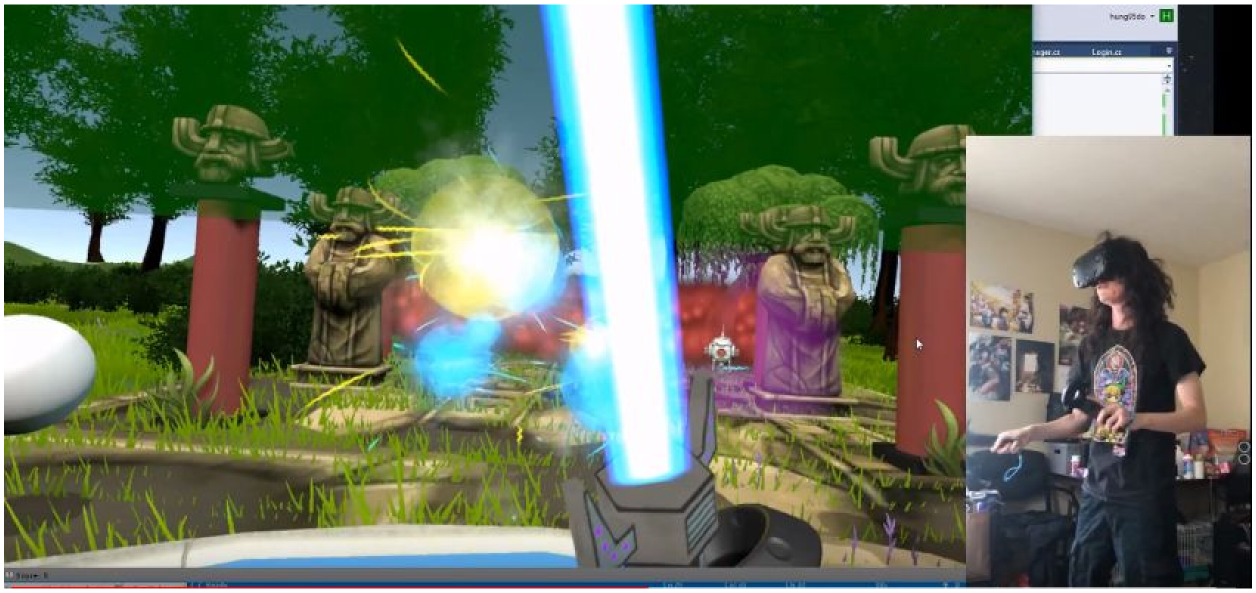
Milestones

Details	Due Date	Status
1.First draft of the game design documentation - whole team	February 29	Done
1. Setup Github - Justice 2. Create Scene and beat object spawner - Ani, Carmen 3. Setting up VR environment - Hung 4. Asset Download - whole team	March 5	Done
1. Revised game design documentation - whole team 2. Create Game menu - Carmen	March 12	Done

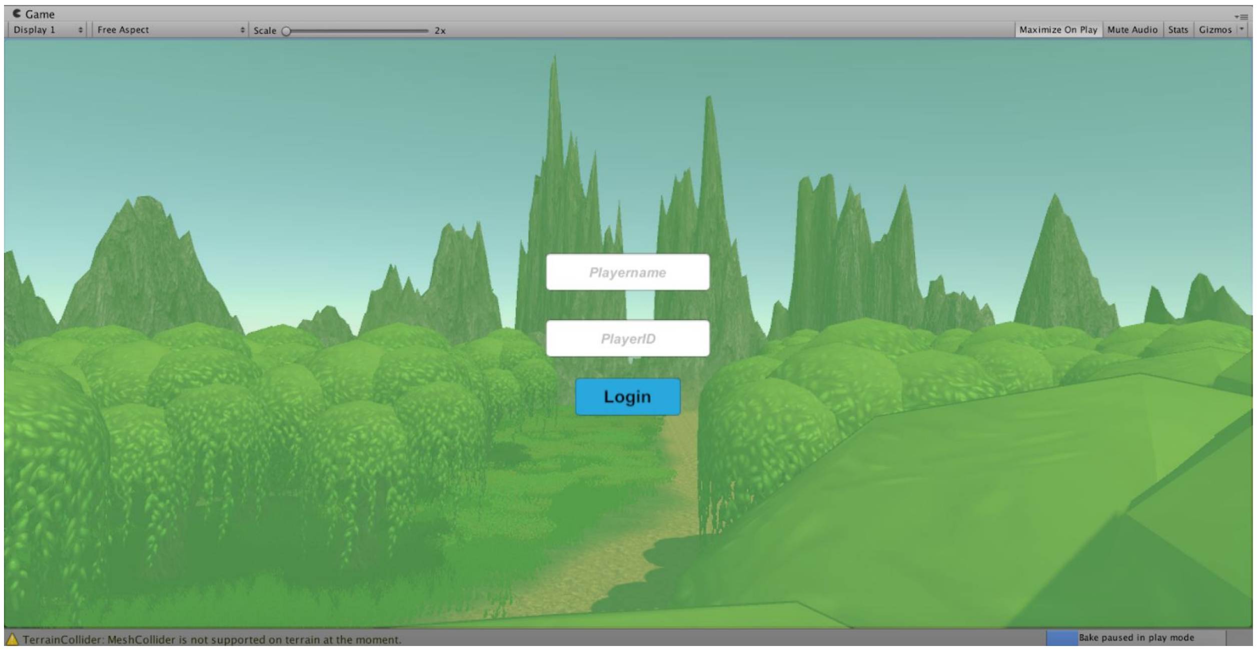
1. Implement robot - Taylor 2. Create the reaction area - Tatem	March 19	Done
1. Example song - Audio team 2. Implement lightsabers - Hung 3. Setup database - Saengduean	March 26	Done
1. Vive controller and trackers - Justice, Hung 2. Implementing beat object and reaction area interaction - Tatem, Justice 3. Switch from SQLite to Firebase and have it works, create table, load and save all user's data - Saengduean 4. Complete song – Audio team	April 16	Done
1.Implement game difficulty adaptivity- Tatem, Taylor 2. Produce audio assets for menu interaction, lightsaber movement, flying robot actions, beat object interaction and accuracy feedback - Audio Team	April 23	Done
1. Debug session and polish the game	April 27	Done
1. Release final product	May 1	Done
1. Final Presentation	May 15	Done

UI ScreenShots



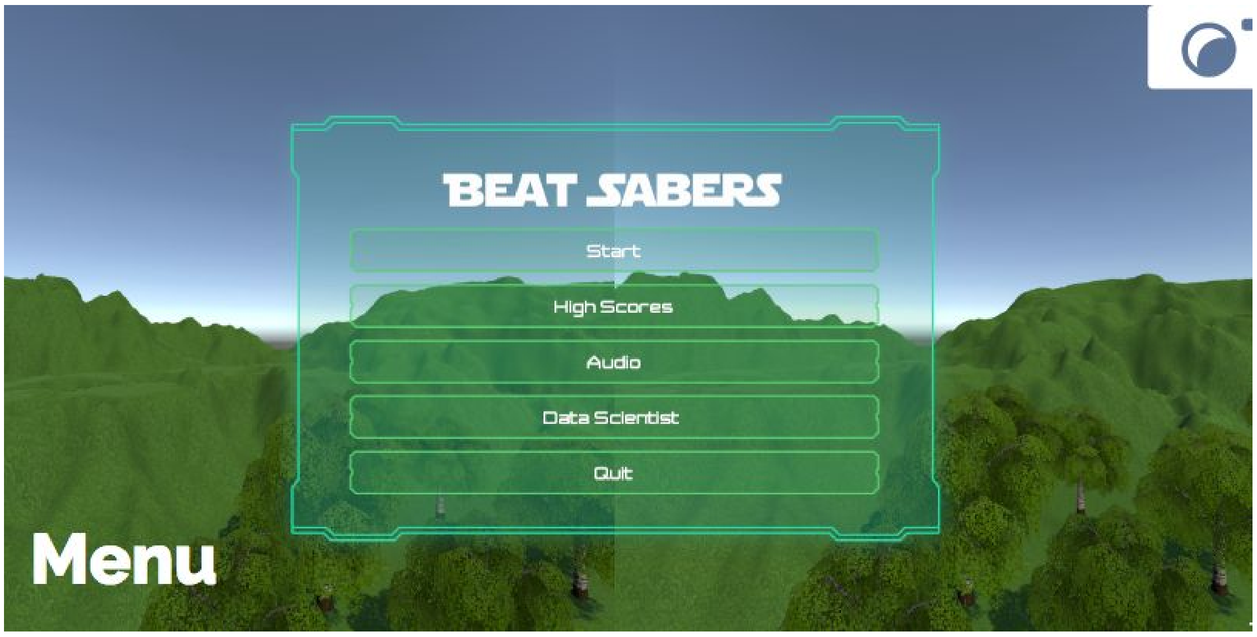


Login:

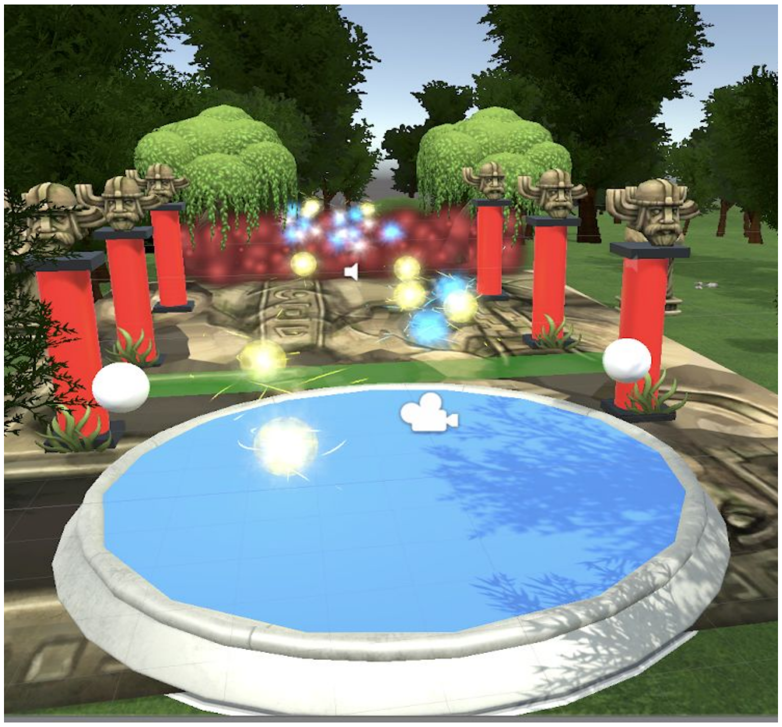


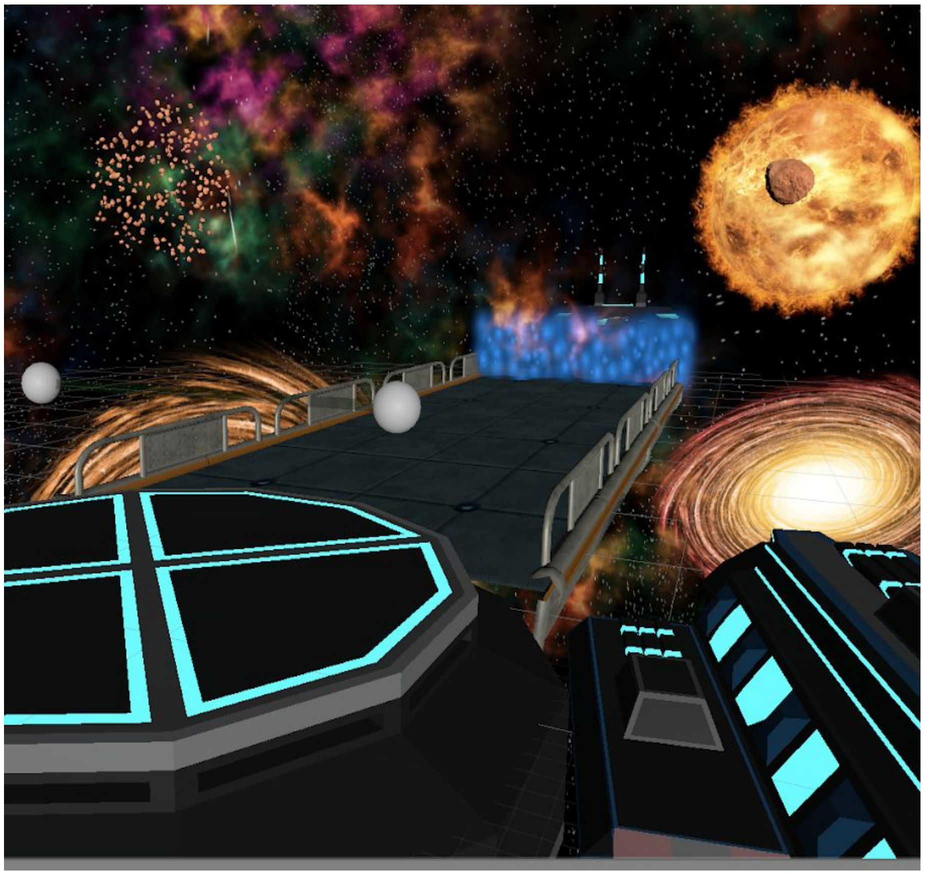
TerrainCollider: MeshCollider is not supported on terrain at the moment.

Menu:



Stages:





Coding Architecture

PlayerProgessHolder: Empty scene that creates object that holds player data and automatically redirects to the Login scene. Key game objects are:

PlayerDriver: Holds scripts with playerData

PlayerProgressHolder: sets initial values for the static playerData class, sets the object to be persistent, and is used to call the SaveData scripts during save and load.

playerData: Stores data all the values explained under the Player table in the Database section.

SaveData: Contains the static playerData class. The function CreateNewPlayerData() creates a new .csv based on a passed string value. The function LoadPlayerData loads the most recent data for a player from the database into the playerData structure. The LoadCreatedGameData function does the same for the scientist file. The SavePlayerData function outputs the data from the playerData to the player's csv file. The SaveCreatedGameData function does the same for the scientist file. Also contains a CreateGame class.

Login: Scene that contains a GUI for the scientist to enter the player's name and ID. Key game objects are:

Canvas: Gui that allows the scientist to enter the playerName and playerID from the keyboard.

LoginObject: processes the data entered in the canvas. Key script is:

Login: Checks for keyboard input and checks for valid login information. The Update() function changes the input field in focus if the Tab button is pressed, calls the LoginButton if the return button is pressed while a playerID is not null, and updates the playerName and playerID based on the values entered. The LoginButton() function sets the playerName and PlayerID in the playerData class to the ones in the scene, if there is a player file with the PlayerName the load function in PlayerProgressHolder is called, the data loaded. If the data matches the input the NewMenu scene is loaded, otherwise an error message is displayed until the input is correct. If the file does not exist, the GenerateNewFile function is called and the NewMenu scene is loaded. The GenerateNewFile function calls the SaveData script's CreateNewPlayerData function to create that file.

NewMenu: Second GUI that controls various aspects of the game. Scientist can set the song and stage from this scene as well as start the main game and quit the game. The GUI is contolled by the mouse. Key game objects are:

<u>CreateGameObject</u><u>:</u> Holds the data for which scene and song to load. As well as the index and tempo of the song to be loaded. By default, it loads the Forest scene and Meltdown song. Key scripts include:

<u>CreateGameData</u>: class that holds the map to be played, name of the song to be played, song index of the song in the AudioManager object, and the BPM of the song.

<u>CreateGame</u>: Contains functions to set the map, nameOfSong, songIndex, and bpm data fields of the CreateGameData in SaveData. The StartButton function checks that the Scientist file exists, creates the file if necessary, and loads the map selected. On load, it sets the values to the Forest scene, Meltdown song, index to 0 and bpm to 165.

<u>Canvas:</u> The start button loads the next scene based on which stage is selected. The Data Scientist scenes loads up another panel that allows the data scientist to either change the audio settings or song, Selecting the stage button will allow scientist to pick either of the songs in the game. After selecting a song, the scientist can then choose either the Forest stage or the Space Stage for the song to be played on. The quit button exits the application.

Forest/Space Scene: Main scenes of the game. Both scenes have identical functions but contain different terrains and placement. Here the player plays the game by striking color-coded orbs with light sabers in time with the song while also dodging lasers from robots that periodically spawn. This lasts until the song ends at which point the player is sent back to the NewMenu scene to either play or new song or quit. This scenes requires the use of an HTC Vive VR set with trackers. Key game objects include:

[CameraRig]: SteamVR object that handles VR tracking and rendering the scene. Additional functionality added to the object include:

EnergySword: Lightsabers attached to the tracker objects that the player uses to destroy orbs of the same color. The left lightsaber is yellow, and the right lightsaber is blue. The left lightsaber has the additional function of pausing and unpausing the game if the touchpad button is pressed and quitting by pressing the trigger while holding down the touchpad button.Key scripts include:

<u>Pausing</u>: Only attached to the left lightsaber. This script has the function of pausing and unpausing the game. The update functions checks if the trigger button is pressed. If the game is paused Unpause is called and if the game is not paused, Pause() is called. If the trigger button is held, the game is paused and the touchpad button is pressed, the song ends and the NewMenu is loaded.
<u>SaberNoise</u>: Plays the stroke sound effect if the lightsaber reaches a velocity high enough to destroy the orbs.

ReactionArea: A set of lines that are used to indicate the proper area to hit the orbs. Also contained in the reaction area is the Spawner object that is centered at the same location. Key script includes:

<u>DrawLine</u>: Draws the line that indicates the valid region of the ReactionArea. The Update() function draws a ling using a lerp where the start point and end point are based on the location of the orbs on either end of the ReactionArea.

Spawner: Spawns the orbs a varying distance behind the ReactionArea based on the tempo of the song being played, the movement speed of the orbs, and the number of beats the song and spawn track are offset by. Which Sequencer calls each object is determined by song choice. Key Scripts include:

<u>Spawner:</u> Determines the location to spawn the orbs. Spawns notes when the SpawnNote function is called by the NoteSpawnSequencer attached to the Audio object. The SpawnNote function determines the distance from spawner an orb is spawned along the z-axis by converting beats per minute to beats per second and multiplying that value by the number of beats the song and spawn track are offset and the movement speed of the orbs at the time of spawning. The orbs are spawned in one of two vertical lanes determined at random. The orbs are spawned in one of seven horizontal lanes. This is partially random, but with the constraints that the value must be within one lane left or right of the last orb spawned of the same color and within 3 lanes left or right of the last orb of the other color. The orb is spawned at the determined location and the numNotesSpawned value in the DifficultyManager is incremented. The spawned orb then has its speed modifier set to the current modifier value. The UpdateMoveSpeed value is used to set the modifier. This value is also used to update the current z value notes are spawned at.

DifficultyManager: This script tracks the players performance and determines how to modify the game based on that data. The update function starts setting the precision value once numNotesSpawned is above zero to the number of notes hit divided by the number spawned. Every 50 notes spawned, the script calls the ManageDifficulty function. It also checks the GameEnded Boolean, if true the EndGame script is called. The ManageDifficulty function checks if the player has hit above 60% of notes spawned or below 80% of notes spawned. If they have hit below 60%, the notes spawned after that point move at speed one unit slower than they did previously, with a lower bound of 1. If they have hit above 80%, the orbs move faster by the same amount with no upper bound. The EndGame function syncs the values in the DifficultyManager with the PlayerData and makes it save that data to the .csv file, then loads the NewMenu scene again. IncrementNotesSpawned increments numNotesSpawned. GetModifier returns the modifier value. BreakStreak checks if the current streak is larger than the max streak, and if it is the max streak is set to the current streak. The current streak is then set to zero and the number of misses is increased. IncrementLasersFailed and IncrementLaserDodged increment the number of lasers the player has failed to dodge and the number of dodged, respectively. Getting hit decreases the score by seven and dodging increases the score by five. IncrementNotesHit uses the passed distance to determine the score. If the absolute value of the distance is less than 0.2 the score is increased by 5 and the perfect hit count is increased by one. If the distance is between 0.2 and 0.5, the score is increased by 2. If the original distance is positive, the early hit count is incremented. If the value is negative the late hit count is incremented. Larger than 0.5 increases score by 1. The note hit count and current streak are then incremented.

RobotSpawner: Handles spawning the robots. Activates when called by the RobotSpawnSequencer attached to the Audio object. Determines the location to spawn the robot randomly in a range between -3 and 4 along the x-axis, between 0 and 5 along the y-axis, and 12 along the z-axis. All these values are floats. The robot is then instantiated.

EndGame: Determines when the game ends when called by the EndGameSequencer attached to the Audio object. SongEnded() calls SetGameEnded in the DifficultyManager script and set that value to true.

Audio: Controls the song and sound effects that occur during gameplay. Also uses AudioHelm sequencers to handle timing of spawning orbs, robots, and ending the song. Each song has its own set of NoteSpawnSequencer, RobotSequencer and EndGameSequencer. Which is active depends on the song. Key scripts include:

AudioManager: When the object is loaded, the Awake() function sets the AudioHelm clocks bpm field, and activates the clock and sequencers that match the song selected. The PlayMusic() function is that called offset by an amount of time based on converting the bpm to beats per second multiplied by the number of beats the song is delayed and divided. Next it adds an AudioSource component to each of the sounds stored in Sounds, OrbSounds, and Music arrays.

Sounds: Class that holds each sound clip and set the volume and pitch at random within a 0 to 1 range and a 0.1 to 3 range as floats, respectively.

Glowing Orb Yellow/Blue: The two types of orbs for the player to destroy. They can only be destroyed by the lightsaber of matching color. They are destroyed if not hit after six seconds. Key scripts include:

SaberCollision(Right): Script that handles collision with the lightsabers. They only detect collision with lightsabers that match their color. The update function tracks the length of time since an orb was spawned and checks if a valid collision has occurred. A valid collision is defined as a lightsaber colliding with an orb of the correct color while the orb is touching the ReactionArea and is moving at square velocity of at least 1. If such a collision occurs, the distance between the center of the orb and the center of the ReactionArea is calculated and passed to the DifficultyManger's IncrementNotesHit function to update the orb. The corresponding tracker then vibrates and a sound effect is played to denote the successful hit. The orb is then destroyed. The OnTriggerEnter function checks if there is collision with the corresponding lightsaber, a Boolean value, Collision, is set to true to denote that the lightsaber is within the orb. The same occurs with the inReactionArea Boolean if the orb collides with the ReactionArea. Collision with a hitbox just before the reaction area causes the orbs to gain a white glow. Collision with the reactionArea additionally changes the glow to black. The OnTriggerExit function sets the inReactionArea and Collided Boolean values to false when the collisions with the respective object end. Additionally, the orb ending its collision with the ReactionArea ends the current note streak with the BreakStreak function in DifficultyManager script.

Cubemove: Controls the movement of the orbs, name comes from an earlier build. The orb is destroyed six seconds after Start(). Update translates the position of the orb along the z-axis by a base value of 9.0 units, reduced by a modifier value determined by player performance, and multiplied by time elapsed. SetModifier sets the value of the modifier based on a passed value.

RoboParent: Robot enemy that when spawned shoots a laser towards the player that the player must dodge. It alerts the player that it has been spawned by making a distinct noise. . Key Scripts are:

RobotFadeIn: Handles fading in and out. On Start, FadeIn coroutine is activated. FadeIn causes the robot to go from transparent to opaque using a lerp and sets the HasFinishedSpawning Boolean in ShootLaser to true when completed. The FadeOut coroutine causes the robot to go from opaque to transparent and calls LaserDodged in LaserCollision when finished. BeginFadeout starts the FadeOut coroutine.

ShootLaser: Controls the movement of the laser. On start, the laser aims at the player. The laser is then moved directly forward relative to the direction the robot is facing. A line illuminates the physical object and moved along with it. After six seconds the FadeOut() function is called and calls the BeginFadeout function in RobotFadeIn.
LaserCollision: Detects if the laser has collided with the player with the OnTriggerEnter function. The collision is determined by the object tagged as MainCamera, which in this case is the Camera (eye) object within the CameraRig. If such a collision has occurred, the player's score decreases by the DifficultyManager's IncrementLasersFailed function. The laserDodged function checks if the laser has previously collided with the player, the player's score is increased by the DifficultyManager's IncrementLasersDodged function. The laser is then destroyed.

File Organization:

All relevant files were in the Assets folder. The folders within contain:

_Scenes: Contains the scenes for the project.
Audio: Contains the songs, midi tracks that are used for the AudioHelm sequencers, and the sound effects.
Material: Contains material files used for environment objects that were not from Unity Asset Store.
Prefabs: Contained are the prefabs that are spawned as orbs and robots during runtime. These are the Glowing orb yellow, Glowing orb blue, and RoboParent. Additionally, there are prefabs of the Audio, ReactionArea, CreateGameObject, and Spawner objects to keep those objects consistent across scenes. Environment folder within contains a copy of the terrain from each scene.
Resources: Contains the database files. Scientist_CreateGame is the scientist file, while the Game_Data_* files contain the player files as detailed under the database section.
Scripts: Contains the scripts used in the game. The base folder contains the files that don't fit cleanly into one of the subfolders. These are the DifficultyManager, DrawLine, EndGame, SaberCollision, SaberCollisionRight, and Spawner Scripts. Subfolders Include:

<u>Audio Scripts</u>: Contains scripts related to the audio. These are the AudioManger and Sounds scripts.
<u>Login and Menu Scripts</u>: Contains scripts used in the Login and Menu scenes. These are the CreateGame, CreateGameData, LoadNextScene, LoadonClick, Login, Player, PlayerData, PlayerProgressHolder, QuitonClick, SaveData, VRUIInput, VRUIItem scripts.
<u>Orb Scripts</u>: Contains scripts that effect the orbs. These are the changeMaterial, cubemove, and onMouseClick scripts.
<u>Robot Scripts</u>: Contains scripts that affect the robots. These are the LaserCollision, RobotFadeIn, RobotSpawner, and
ShootLaser scripts.
<u>Saber Scripts</u>: Contains scripts that affect the Lightsabers. These are the Pausing, SaberNoise, SaberNoiseRight, and swordswing scripts.

Asset store file folders:

AudioHelm: Contain files for AudioHelm plugin. Midi sequencers are used in the project to handle spawning of orbs, robots, and ending the stage when a song is over.
EnergySwords: Contains models and original prefabs for the lightsabers.
FantasyEnvironments: Contains model for the terrain in the NewMenu scene.
Fonts: Contains fonts used in NewMenu scene
Forest: Contains models for the terrain used in the Forest scene.
Glowing orbs Pack: Contain models and original prefabs used for the orbs.
NatureStarterKit2: Contains models for the terrain used in the Login scene.
Sample UI: Contains UI prefabs used in NewMenu scene.
SciFi_Space_Drone: Contains model, animation, and original prefab used for the RoboParent.
Simple Health Bar: Unused in final build.
SKY city lite: Contains model for the platform used to denote the play area in the Forest scene.
Standard Assets (Mobile): Unused in final build.
SteamVR: Contains files for SteamVR plugin. Included are the scripts, models, prefabs and other files used to interact with the HTC Vive.

Tree 1_Textures: Unused in final build.
Tree Textures: Unused in final build.
Vast Outer Space: Contains models and textures for the terrain used in the Space scene.
Warzone: Unused in final build.

Database

All the data stored in Comma Separated Values or CSV file. The player's file generate base on their name. The scientist's file stored the data for all players. All the files can be found in Resources folder under Assets.

Table:

Player	
playerName	string
playerID	string
currentSong	string
timeStamp	time
score	int
earlyHit	int
perfectHit	int
lateHit	int
miss	int
currentStreak	int
noOfObsSpawning	int
precision	float
map	string

playerName - player's name

playerID - the player's ID that will be given by UCSF

currentSong - name of the song played

timeStamp - time of note in song

score - total score the player accumulates over the course of the song

earlyHit - total number of notes hit within the reaction area, but too early to be a perfect hit

perfectHit - total number of notes hit within a very close to the center of the reaction area

lateHit - total number of notes hit within the reaction area, but too late to be a perfect hit

Miss - total number of notes that are not hit at all

currentStreak - consecutive number of notes hit

precision - percent of notes hit over number of notes spawned

map - stage selected

ScientistData	
playerName	string
nameOfSong	stirng
map	string

playerName - name of the player

nameOfSong - name of song played

Map - stage selected

Problems and RoadBlocks

We had a couple of problems and roadblocks when dealing with this game. We had some hardware and software issues.

For hardware, the struggles were involving the VR console. Reflecting on the project, one thing that could have improved our productivity would have been having the headset earlier. Early in the development, our team stagnated a bit due to not knowing how to work with the VR equipment we had. Ultimately, we discovered that we needed a vive headset, rather than the given gear VR. We had hardcode controls with the keyboard and mouse until we were able to acquire it. After we got the headset there was a good workflow that would get interrupted due to having to share the headset, so it was hard to take breaks during development to wait for the equipment to be available.

As for the software aspect, Unity was something new for all us and there was large portion of time dedicated to learn the application. Each of us took over different aspects of Unity to learn based on our assignments for the game. So there were constant miscommunication, roadblocks of not being able to move forward and integration problem of mixing the code with VR.

Other than those issues the teamwork and overall game development was successful and we emerged with a very fun and exciting virtual reality experience.

Future Goals

- Stomp feature that destroys all the orbs in its path. As you gain points, there will also be power up bars getting filled. Once filled, the player can stomp his foot and majority of the objects that is facing the corresponding side of the foot will disappear. So if the player stomps his left foot, majority of the objects from the left side will disappear. The stomp action can also be used to destroy the robot immediately. This is will provide extra points to the player. The music need no be intuned with the foot stomp as it depends on the player's ability and other power ups.
- In order to bring the longevity for the game, there is going a story that is told in three chapters. Each chapter is set in different stages and have different tunes through the game. We initially started with a storyline but did not have enough time to go through it and add it to the game. This is what we had for the possible storyline :

  "In a place far far away Azurlia a virtual reality haven has been corrupted by three sinister villains, they call themselves Vorbax. They used brute force to infiltrate Azurlia's main source of power which are based in the temple of Melodi. Ever since that fateful day, the virtual realm has never been the same stricken with misery, turmoil and strife. The Lightsabers are an ancient counsel that protects all from mischievous beings. Seeing this horrific distress they have chosen a hidden champion and that champion is you. Since you are given this arduous task you must save Azurlia and restore the balance before it is too late and all is lost. Enter Azurlia the virtual realm and defeat the three evils, they will try to break you down but you must persist onwards for the sake of the people. You are the last hope for Azurlia."
- Have an adaptivity aspect were player will not exactly know when the transition happens, it will change automatically based on progress of the player. The adaptivity of the music

shall be by the increase and decrease of the number of beats that works on one tune. For Example, If the player hits 25 perfect hits the beats increases to the same tune. If the player starts missing, the beats decreases and go back to easy.

- Add more stages and enemies
- Game unlockables and maybe a possible double-edge lightsabers
- Create scoreboard to show the player's stats.

Conclusion

Finally, this is a special game for the development team and audio team as all of us were very excited to create BeatSabers even though the majority of us had never worked on creating a game before. BeatSabers was done for players to have fun, embrace the Star Wars fandom and create an impact for the scientific community in order to collect data for cognitive brain function. All of us are very proud of what we were able to accomplish with this game because we were able to complete nearly everything we wanted to before the end of the semester. We hope you enjoy the game as much as we did in creating it and MAY THE FORCE BE WITH YOU.