PROJECT REPORT

Collaborators:

1- Zeina Khadem (znk217@nyu.edu)

2-Akshith Karthik (ak10747@nyu.edu)

3- Godbless Osei (gmo6996@nyu.edu)

1, Introduction:

Many people love going to live concerts for the fun and excitement they bring. However, not everyone can join in. Some fans have health issues that make it risky to be in big crowds or around loud noises. "Build Your Concert" is a video game project that lets you enjoy a concert without having to worry about these risks. It's perfect for fans who have to be careful about their health, or for anyone who finds flashing concert lights and loud sounds too much to handle. With this game, we want to make sure everyone can have the experience of a live concert in a way that's safe and comfortable for them. It's all about bringing the joy of music to more people, using technology to make concerts accessible to everyone, everywhere.

An article published by the US National Center for Biotechnology Information studied the effects of stroboscopic lights during EDM concerts in Amsterdam, and concluded that these lights triple the threat of epileptic seizures in susceptible individuals.

Additionally, there are a plethora of external factors that affect the ability of people being able to get out of their house and attend social events with entertainment value, such as a concert. A very recent and relevant example being that of people that were confined to their homes due to the COVID 19 pandemic. A study that reflected the mental health condition of home confined young

adults during the pandemic reflected a very prominent positive relationship between the lack of social interaction and deteriorating mental health and happiness, which in turn has adverse effects on the society as a whole.

Our project aims at introducing some level of inclusivity for people that have certain challenges or difficulties in attending live concerts and enjoying the feel of being at an actual concert. We wanted to simulate the experience of being at a concert, at the comfort of your house with the tap of a button.

2. Project Development:

The process of developing the project was spread out over the course of the term, we spent the first couple of weeks until the submission of the project idea proposal brainstorming a real world problem that we could effectively counter using the VR domain and the significance of the simulation that we are creating to that particular problem. We settled on creating the virtual concert for people that have challenges leaving their homes to enjoy concerts as well as people that are unable to attend concerts due to issues such as photosensitive epilepsy brought on by flashing lights. The proposal essentially served as an intersection between two domains that we were encouraged to work under, i.e, entertainment and healthcare.

Once we had the scope of the project laid out, we shifted focus on creating a concert environment that suits our project. This is when we started working on Unity, we started off on a plain project, obtaining elements and assets that we found necessary for our concert scene, some of the assets were downloaded from the Unity Asset Store while others were custom built by us

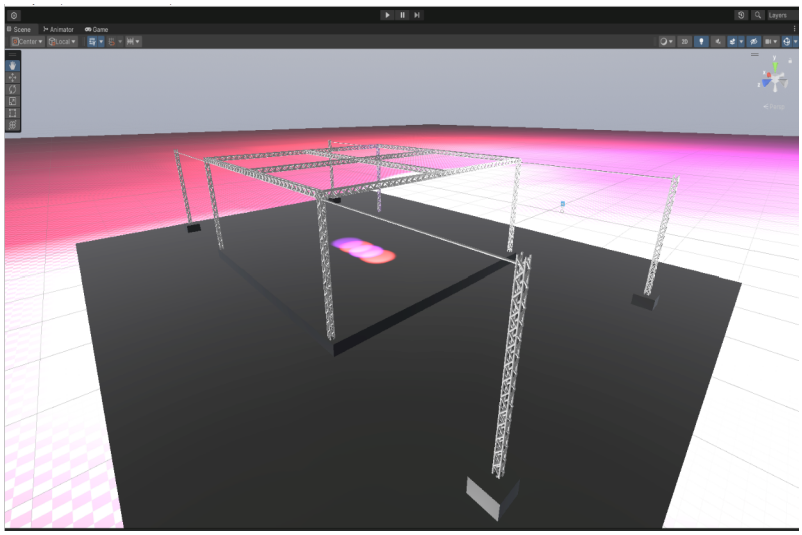
using the 3D shapes that are built into the Unity interface, such as cubes and cylinders. We used a combination of these to create a baseline environment that included a stage with curtains, a DJ setup, speakers, truss systems and light holder components as well as an enclosing wall, some high definition 3D characters for the crowd, performer, security, and some miscellaneous stands in order for our scene to closely resemble a real concert.

ASSETS USED:

Speakers:

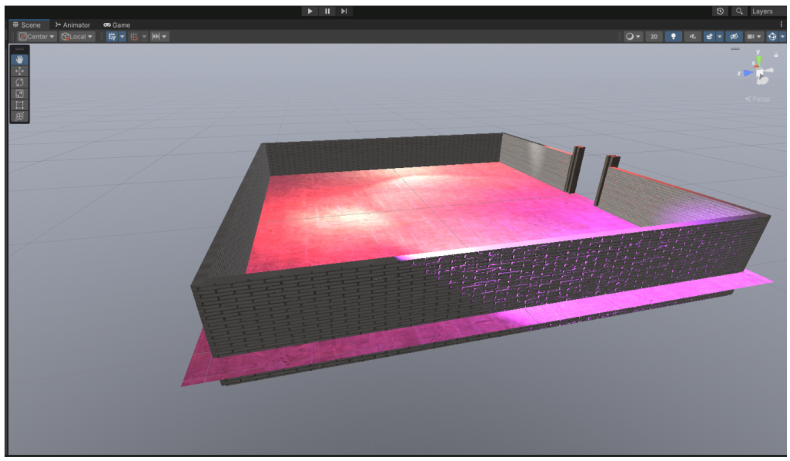
These Speakers were downloaded from Unity Assets Store and imported into the project. They were attached to the truss system to better mimic a real world concert.

Truss System:



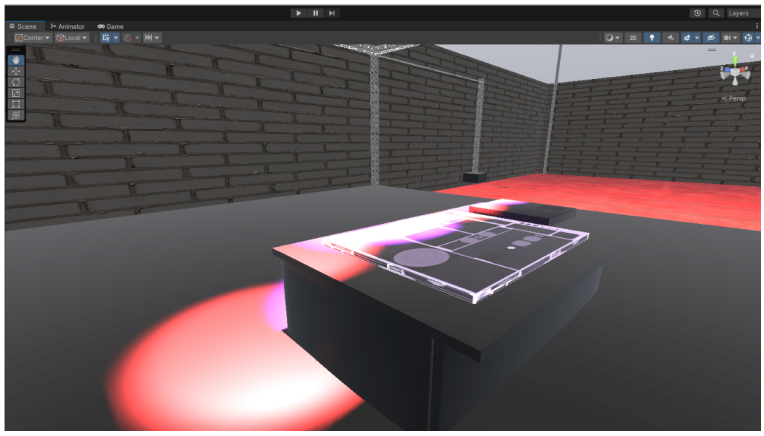
The truss system was made with cylinders. First we made four cylinders to serve as the Strut of the truss and then arranged a number of smaller cylinders in a zig-zag fashion to form the bracings of the truss. We confined these parts to form a replica of a Pratt truss system.

Walls:



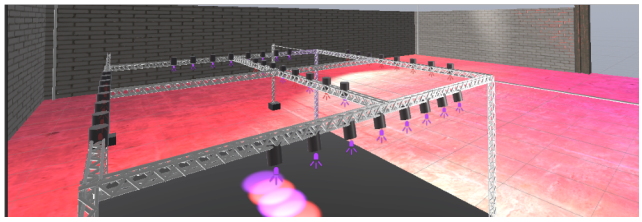
The walls of the concert were made with cubes found in Unity. A brick-like material downloaded from Unity and attached to the cube.

DJ Setup:



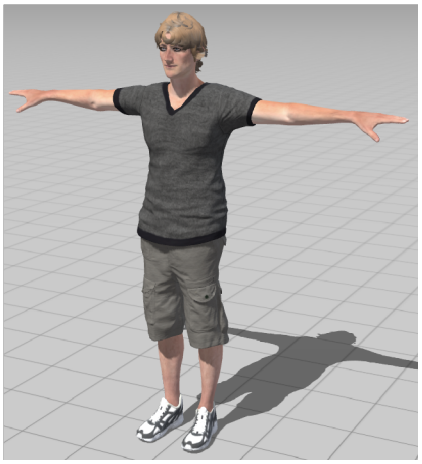
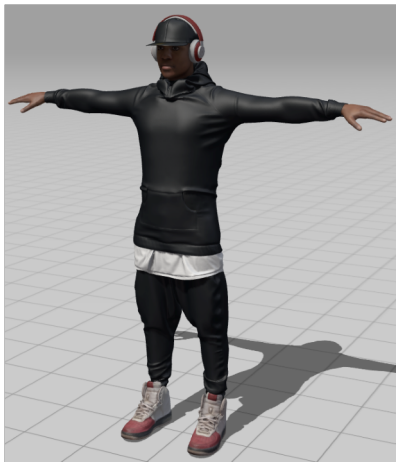
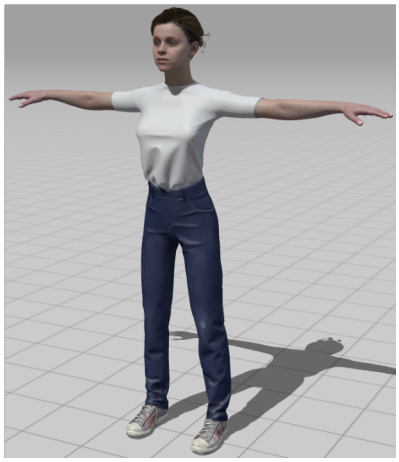
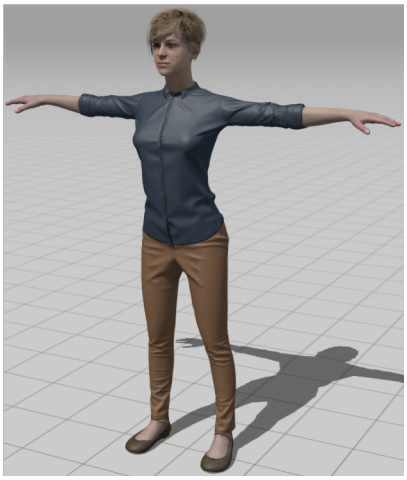
Here we used the cubes to arrange the platform the DJ equipment is placed on. We downloaded a DJ controller from unity and placed it on the platform.

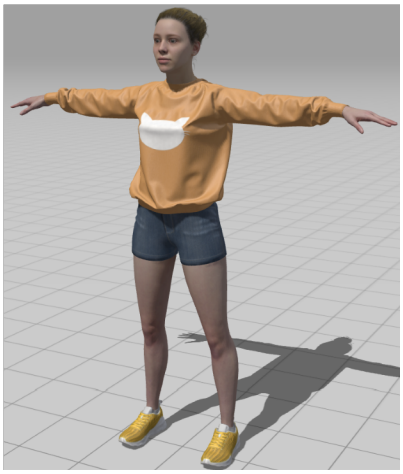
Lighting:

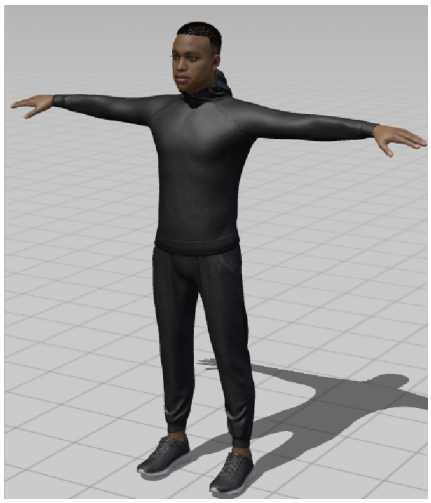


To make the lights we used Unity built in spotlight and point lights. We created a cylinder to serve as a housing for each light created.

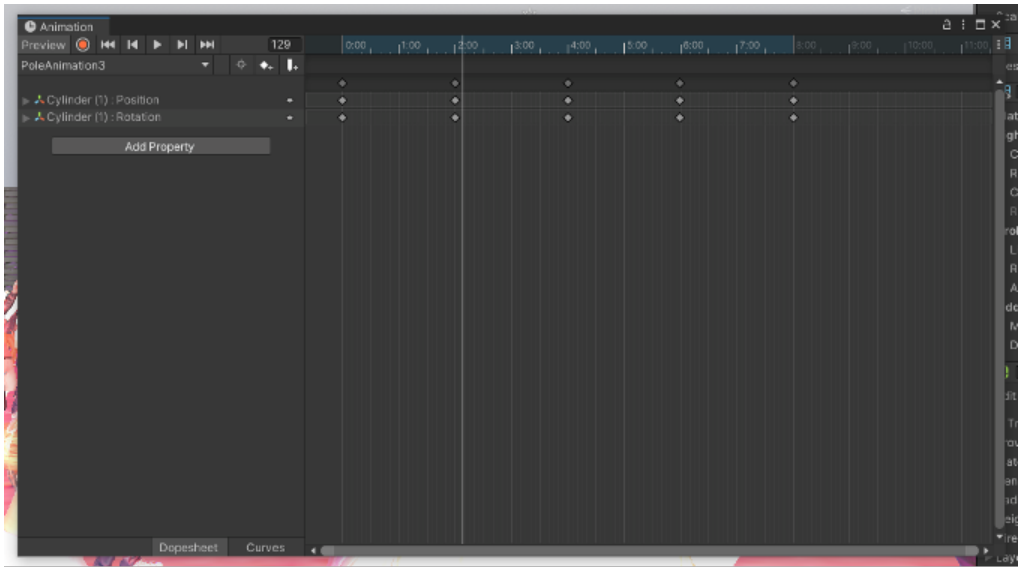
Characters:



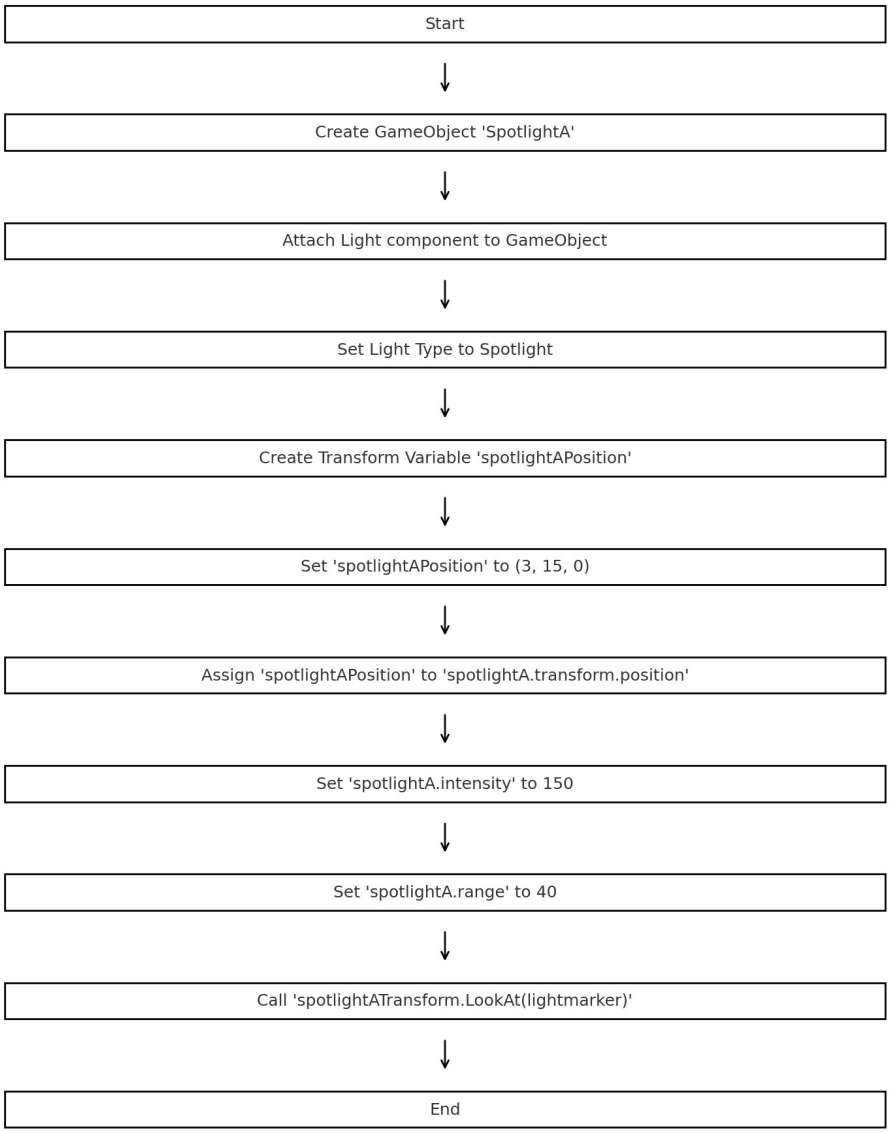



All these characters were downloaded from the Mixamo Assets store and then animated.
Once we had the concert environment set up with our stage and characters, we moved on to adding lights, both stage lights and Stairville Headlights for the arena and animating these lights to move in particular way to add depth and effect to our simulation, we had a mixture of red and purple lights present on the stage using the spot light object that is available on Unity which we attached to cylinders that acted as placeholders for the lights. Next we manipulated the movement of the lights using the built in manual animation function that Unity has. Using the Animation pane, we recorded the light over a period of time to move in a certain direction along a rotational axis, and we set keyframes for distinct positions of the light and Unity uses this information to create a moving animation of the lights moving from one position to the other, and we set this animation to loop. We applied this principle to the lights that are present above the stage and the pole light that lights up the arena from all four sides. The lights and its animation work in unison to brighten the stage and the entire environment, until they need to be turned off.



We also used an alternative approach to animate one of the spotlights which moves along the stage and moves in a custom motion periodically. We worked on this light using C# code which we developed on the Visual Studio Editor.

Flowchart of Light Animation:



C# Script:

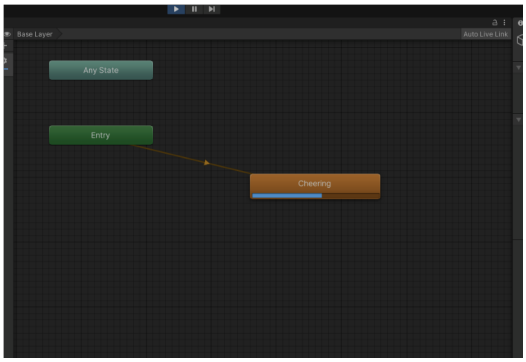
```
using System.Collections; using
System.Collections.Generic;
using Unity.VisualScripting; using
UnityEngine;

public class Lighting : MonoBehaviour
{   //Reference the spotlight marker
    public Transform lightmarker;
```

```
    //Create a variable for pointing to the light
    Transform spotlightATransform;

    // Start is called before the first frame update
    void Start()
    {
        //Make a game object
        GameObject SpotlightA = new GameObject("SpotlightA");

        //Connect the light to look at the mark spotlightATransform
        = SpotlightA.transform;

        //Add light component to the game object
        Light spotlightA = SpotlightA.AddComponent<Light>();

        //Change the light from a point to a spotlight
        spotlightA.type = LightType.Spot;

        //Create a light position variable
        Vector3 spotlightAPosition;

        //Choose a position spotlightAPosition =
        new Vector3(3, 15, 0);


        //Change the position

        spotlightA.transform.position = spotlightAPosition;

        spotlightA.intensity = 150;

        spotlightA.range = 40;
    }
    // Update is called once per frame

    void Update()
    {
        spotlightATransform.LookAt(lightmarker);
    }
}
```

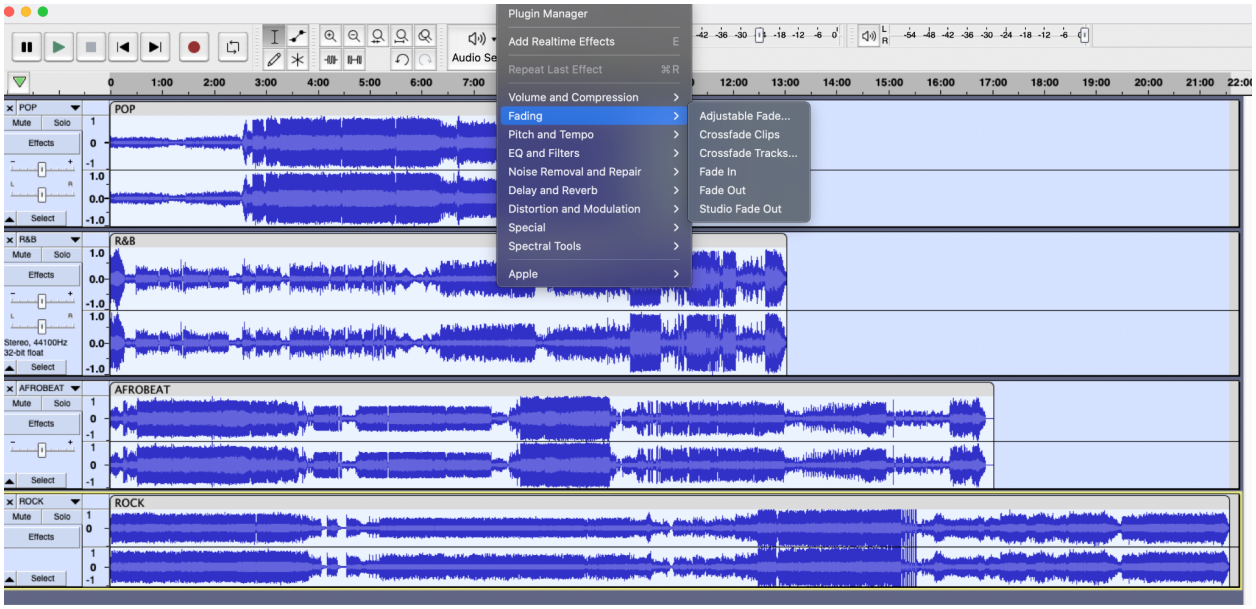
Explanation:

This script creates a class called Lights which inherits from monobehaviour (This allows us to use functions like awake( ), start( ), update( ), and among others). We declare a public data member of the class called lightmark. This lightmark transform stores position and rotation data. We then create a variable called spotLightTranform which points to light. In the start function we create a new game object called spotlight which is activated during runtime. The transform component is added to the spotlight game object and assigned to the spotLightTranform variable. We add the spotlight as a game component by using the Addcomponent keyword keyword. We then set the type light component as a spot light. We create a 3D vector component and assign it the transformation component of the spotlight transformation. Afterwards we set the position of the 3D vector. We change the intensity and range of the spotlight using spotlight.intensity and spotlight.range. In the update function we transform the spotlight to look at lightmark game object which we have created and animated in the game scene. The update function changes the position and rotation of the spotlight after every frame.

With the lights implemented, we moved onto attaching animations to the characters to make the scene look more realistic, we took different cheering and dancing skeletal animations from Mixamo, imported those animations into Unity and created Animation controllers that we assigned the animations to. The animation controller helps us to adjust the features of the animation as well as link the animation to a particular character or asset in the scene. After assigning all the different animation to the animation controllers, we linked all the animation controllers to the characters and set them to loop.



The most important part of a concert is the music, we needed audio that would give the user a feel of being in a concert setting, which is why we used audio from real concert footages for popular songs that were available on Youtube, we additionally integrated crowd sounds to this audio and created a mix of songs for each of the four genres that we had: Afrobeat, Pop, Rock and R&B. The editing process of the audio was done with Audacity to merge the different audio files and integrate into one streamless audio track, to sound as though it was played by a DJ at a concert.

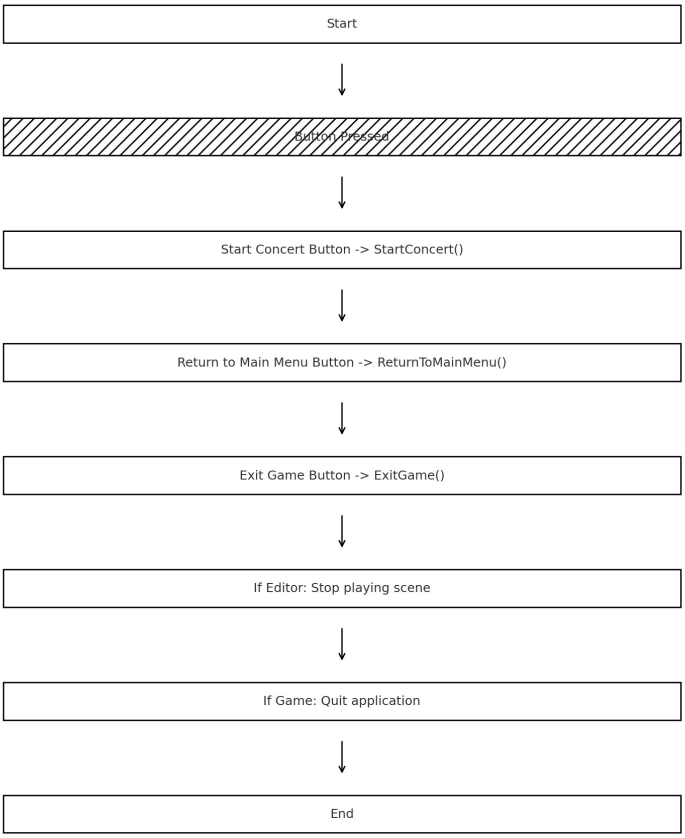


Next we needed to add the audio files into the assets that we had and link them to Game Objects that we can use to change between the audio tracks and let the user change the music. The implementation helped us in creating a script with functions that would be implemented into the menu that allows the user to select the genre that they would like to listen to.

We had two menus, the main menu and an in-game menu. The main menu is the first scene that the user would see when they started the simulation and it gives them the option to Start the Concert or to exit the game. When the user selects the start concert button we teleport them into the concert scene, where they can view another menu that has the different genres to choose from as well as a button that enables them to toggle the lights on and off.

Main Menu:

FlowChart:



C# Code:

```
using UnityEngine; using
UnityEngine.SceneManagement; #if
UNITY_EDITOR
using UnityEditor;
#endif


public class SceneSwitcher : MonoBehaviour {
    // Function to switch scenes by providing the scene name
    void Start()
```

```
{
    // This method is currently empty in this script's implementation.
    // The Start() method is called automatically when the script is first run.

}

// Method to switch to the "Concert" scene when the corresponding button is pressed
public void StartConcert()
{
    // Log a message to the Unity console for debugging purposes
    Debug.Log("Pressing");

    // Load the "Concert" scene using Unity's SceneManager
    SceneManager.LoadScene("Concert");
}

// Method to switch to the "MainMenu2" scene when the corresponding button is pressed
public void ReturntoMainMenu()
{
    // Load the "MainMenu2" scene using Unity's SceneManager
    SceneManager.LoadScene("MainMenu2");
}
// Method to exit the game when the corresponding button is pressed
public void ExitGame()
```

```
    {
        // Log a message to the Unity console indicating the quit request
        Debug.Log("Quit requested");

        // Check if the game is running in the Unity Editor
#if UNITY_EDITOR
        // If running in the Unity Editor, stop playing the scene
        EditorApplication.isPlaying = false;
#else
        // If running outside the Unity Editor, quit the application
        Application.Quit();
#endif
    }
}
```

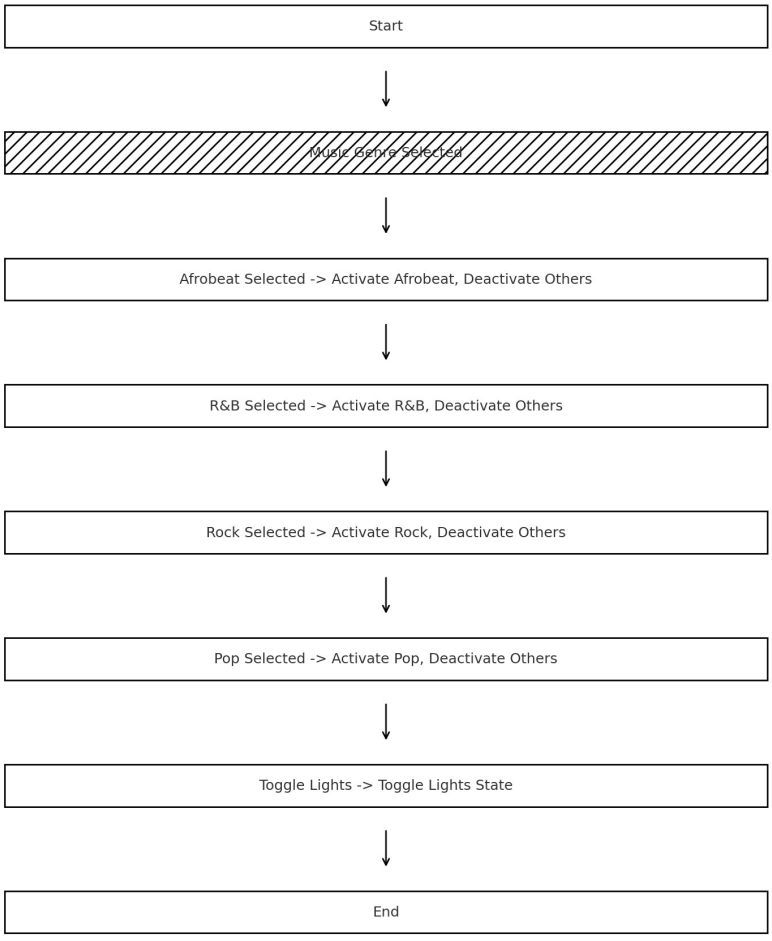
Explanation:

This script primarily uses the LoadScene function of Unity which is a teleportation function between one scene to the other. Initially the code checks if the program is using the Unity Editor, then it declares the Public Class of our main script with MonoBehaviour which contains all the preset functions such as the LoadScene we are using. Then it creates a function to load the concert scene and uses LoadScene to teleport to the concert Scene. It also sends a message to the Unity Console which we used for debugging purposes. It also creates another function called the ExitGame function which first checks if the program is running in the editor, if it is then it sets

the status of the EditorApplication to false, if is it not running in the editor it exits the progam using Application.Quit.

Concert Menu

Flowchart:



C# Code:

```
using UnityEngine;
```

```
public class MusicController : MonoBehaviour
{
    // Declare public GameObjects for each music genre. public GameObject
    Afrobeat; public GameObject RB; // Use 'RB' instead of 'R&B' to avoid
    syntax issues.
    public GameObject Rock;
    public GameObject Pop;

    // Declare public GameObject for lights.
    public GameObject Lights;

    // Call this function to play Afrobeat music.
    // It activates the Afrobeat object and deactivates others.
    public void PlayAfrobeat()
    {
        Afrobeat.SetActive(true);
        RB.SetActive(false);
        Rock.SetActive(false);
        Pop.SetActive(false);
    }
    // Call this function to play R&B music. // It activates
    the R&B object and deactivates others. public void
    PlayRB()
    {
```

```
    Afrobeat.SetActive(false);
    RB.SetActive(true);
    Rock.SetActive(false);
    Pop.SetActive(false);
}

// Call this function to play Rock music. // It activates
the Rock object and deactivates others. public void
PlayRock()
{
    Afrobeat.SetActive(false);
    RB.SetActive(false);
    Rock.SetActive(true);
    Pop.SetActive(false);
}

// Call this function to play Pop music. // It activates
the Pop object and deactivates others. public void
PlayPop()
{
    Afrobeat.SetActive(false);
    RB.SetActive(false);
    Rock.SetActive(false);
    Pop.SetActive(true);
}
```

```
    // Toggle the active state of the Lights object.
    public void ToggleLights()
    {
        // If the Lights object is active, deactivate it, and vice versa.
        Lights.SetActive(!Lights.activeSelf);
    }
}
```

Explanation:

This script declares all the game objects that holds the audio files with the different genres of music. Then it creates a function for each of the music genres, for example the public void PlayAfrobeat function, takes the status of the game object called afrobeat and sets the status to true, which means that it is made active and therefore that audio will play and sets the status of all the other audio game objects to false and therefore deactivates them. It does the same for all the other genres, creating different functions for each genre that activates that specific genre and deactivates the others so that when that function is attached to its corresponding button on the menu, the desired track will play.

There is also a function to toggle the lights on and off called the ToggleLights() function, which uses a preset function called Lights.activeSelf that returns the status of the light, whether it is active or not active. This is returned as a Bool value true or false, when the status is returned, the lights set active function will invert the status of the lights.

3. Results and Evaluation:

Functionality Reasoning

Once we had a fully functioning simulation, we moved onto the testing and evaluation stage of the project, we wanted to assess the functionality of the project and what we took into primary consideration was the ease of operation for the user as well as how accurately the project complies with the logic and real world comparison of the project.

We initially considered implementing a feature to enable the user to be able to walk around and explore the concert environment, including going close to the stage and taking a closer look at the elements of the stage or walking by the side to explore stands, however we encountered a logical dilemma, where we wondered if this would directly translate to a real world setting as in a real life concert, being packed amongst a crowd, it would not be possible for a person to move around or go closer to the stage, it was at this point that we rather opted to add security guards and a set of barricades separating the crowd from the stage.

We also curated the menu to suit ease of functionality, keeping the in-concert menu always available to the user even after they had selected the audio track that they wish to play, so that they still have the option to switch to a different genre if they wish to, as well as access the light toggle function at any point in the concert if it causes them discomfort later on. We achieved this primarily with the design of our code, which activates the game object that holds an audio file to be played when the corresponding button is selected while all other other game objects that hold the other genres are deactivated. Choosing a different genre in the middle of playing the game will switch the status of the game object, making the user experience relatively smooth and streamlined.

In addition to the audio option, the Toggle Lights button is also present in the Concert Menu, and we used a single function in our code to turn the lights on and off, by simply making the program

to return the current state of the game object and flip its status when the button is interacted with, rather than implementing two different buttons to turn the lights on and off. This aided in improving our functionality.

We also implemented a return to main menu option with ease of use in mind as it enables the user to exit the game from the main menu when they're done with the concert experience.

Testing:

Some of the tests we ran on our project after building it onto the Oculus Quest 2 and the changes we made from our observations included:

1. Accessing the Concert Scene from the Main Menu that shows when the game runs: This transition to the scene was smooth, however initially the alignment of the menu in the scene was not well suited to enable the user to select an option easily, and the options were not clearly visible. We later adjusted the position of the menu and added some descriptive text that makes it easy for the user to navigate through the menu.
2. Selecting an initial Audio genre and then switching to another: In the concert scene, we started with the Afrobeat genre, selecting the button for that audio to play and after it played for a couple of minutes we pressed the button for the Rock genre to shift to that audio track, we did observe a delay of a couple of seconds during which no audio is playing but this can be attributed to the size of the file and the time the program takes to play the audio file.
3. Toggling the lights on and off: We tested if the lights are being turned on and off multiple times, we tried using the button at the beginning of the game, during the simulation while

an audio file was being played, and even pushing its functionality to its limit by repeatedly toggling the lights on and off. In all situations we observed a relatively smooth operation of the lights function.

4. Moving Player head around to explore the scene: On our simulation, the user should be able to turn around their head and observe the concert scene to see the different implementations put in place, and look at the animated characters that have been placed to their sides, as well as move their body about a fixed position as though they are grooving to the music, within the game environment of Unity, this function can be carried out well, however we did observe that on the Oculus Quest 2, moving around results in a laggy footage of the character animations. Testing the gameplay on the Quest 2 Pro however resulted in much smoother footage, and we were able to identify the issue as the high definition characters which have been duplicated multiple times which consumes a lot of the processing power of the Oculus Quest 2, resulting in the lag.

Challenges

During the implementation process, we faced two major issues that posed a challenge to us in the smooth running and functionality of our simulation. The first of which was controlling the animation of the lights accurately. We had a functioning code that operating the animation of the spotlight that was based on the stage, however we faced a challenge assigning the right positions to the light as well as controlling the speed of the light, which made the light move around too quickly. We were able to fix this issue by combining the hand animation features of Unity along with our code to correct the speed and the positioning. We also observed during file transfers

from one PC to the other, or from our drive to the Lab PC, we would lose some elements of our scene, especially some of the animated lights, which we had to reimplement.

Secondly, after we implemented the Main Menu and the Concert Menu with the buttons, we built the project onto the Oculus and observed that our buttons were not functioning. When the user tried to use the Poke interactive feature on the button, the hand icon would go straight through the button and would not interact with it. We reassessed our code as well as our implementation of the buttons and if we assigned the scripts accurately to the buttons, we also re implemented the VR Interaction using the Oculus website tutorial. However, the issue pertained and we were finally able to fix this by duplicating the template that was provided, and copy all the assets from our scene onto that template scene and reimplemented the buttons and the buttons started to function.

4. Conclusion and Future Work:

Throughout our program, we have managed to achieve a main menu screen which gives you the option to start the concert or to exit. The start concert option takes you to the concert environment which we have built with a stage, spotlights, stage lights as well as the option to turn those lights on or off. We were able to incorporate 4 different music genre options for the user to choose from as well as giving them the option to return to the main menu whenever they decide they're done with their concert experience. Throughout our environment we implemented various aspects that resemble a real life concert, like adding various characters around the user with slightly differing animations and varying looks. Bodyguard characters and barricades near

the stage as well as different color stage lights and a DJ stand on the stage. We have also included stands along the sides to mimic the stands you find in concert arenas.

Despite these achievements in our program, we recognize that there are areas for improvement for our program. For one, we could increase the amount of genres the user can choose from, catering to varying music tastes to maximize user satisfaction. Another aspect that we could implement to elevate our project is by allowing the user to choose different concert arenas. How cool would it be if you could pretend you were attending a concert from Madison Square Garden? The choice between venues further increases the inclusivity of our project because traveling to different countries around the world isn't an option for everyone. Furthermore, another way we could elevate our project is multiplayer options so you can join other people's concerts. It would function almost like a silent disco, allowing you to enjoy your time with your friends and whoever wants to join without putting anyone at risk or without the issue of geographical barriers.

5. Reflection on Learning:

Teamwork and Collaboration

Throughout the duration of this project, a fundamental skill that emerged was the aptitude for effective teamwork. Collaboratively, we learned the art of task distribution, leveraging each other's strengths, and navigating challenges collectively while ensuring sustained morale. The cohesion within the team proved indispensable, particularly when faced with unfamiliar interfaces such as Unity and virtual reality. Despite the novel challenges, our collective effort maintained high morale, fostering an environment conducive to innovation and progress.

Knowledge Exchange and Skill Development

An exemplar manifestation of our collaborative spirit was witnessed in the domain of audio and music within the project. Leveraging Akshith's prior experience with Audacity, a platform for audio editing, knowledge transfer occurred organically. This exchange facilitated the education of Zeina and Godbless in the intricacies of audio editing, showcasing our capacity to learn and adapt from one another. Although our project's coding component was limited, the foray into

CSharp basics and their application in a distinct syntax was both stimulating and enriching.

Insights into Game Development Complexity

While our project might not have extensively involved coding, the exposure to Unity and VR provided a profound appreciation for the intricacies and challenges inherent in game development. Witnessing firsthand the meticulousness, effort, organizational demands, and temporal commitment requisite for game creation augmented our respect for game developers.

Project Management and Task Structuring

Integral to our project's success was the acquisition of project management and organizational skills. Learning to efficiently partition tasks, optimize workflow, and meticulously structure activities underscored the significance of effective task delineation. This approach allowed us to maximize our collective output and attain significant milestones within the project's scope.

Conclusion

The journey through the development of "Build Your Concert" not only facilitated the acquisition of technical skills but also fostered a holistic understanding of collaborative work dynamics and project management. These acquired proficiencies transcend the confines of this

project, equipping us with a robust skill set crucial for future endeavors in technology-driven innovation.

Link to GitHub Repository: https://github.com/kzeina/Build-A-Concert **Link to Google Drive with Project File:**
https://drive.google.com/file/d/1_-2CH1DCkxACLBL3vqTwfMOyuBEgFM3a/view?usp=drive_link

References:

Statistical Data: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6585837/
Models and Assets:
https://www.daz3d.com/customer/account/downloadstudio/?cust_id=8510890
Characters and Animations: https://www.mixamo.com/#/
Audio: https://www.audacityteam.org/

Songs:

Afrobeat;
-Charm by Rema : https://youtu.be/pGQ2ezbwUeE?si=cFReDpNP2e7U9Xio
- Calm Down by Rema: https://youtu.be/yRmiAYnG-Q8?si=lNyxikQFYmIVeOSj
- Ku Lo Sa by Oxlade: https://youtu.be/qOQE-BqqQZU?si=b5_VEPNf81XcGyXr -
Last Last by Burna Boy: https://youtu.be/oFVrNmC6u2Q?si=yVz-XfZcJaUP8LOj
R&B:
-DNA by Kendrick Lamar: https://youtu.be/A4H65_VPGa8?si=z-aYUcvdsAIXtCz7
-HUMBLE. By Kendrick Lamar: https://youtu.be/bI2kzKEyjBI?si=Tsm4xycB03FgbwDh
-Die For You by The Weeknd: https://youtu.be/wX0anflee0U?si=cyDGld8-nhDsSTo7 -
Let Em Know by Bryson Tiller: https://youtu.be/ij_vQo4yGv0?si=VACov-eClSoWP-Iv

Pop:
-Umbrella by Rihanna: https://youtu.be/462PvcFUHR8?si=DSP1XvKLGDmVPNWr -
Diamonds by Rihanna: https://youtu.be/lmPmKezTeiw?si=BuDbjHoPYMC5eG5F -Night Changes by One Direction: https://youtu.be/TtdyYfT2Z3M?si=kT-6g5Q2KEh2U_yy
-Golden by Harry Styles: https://youtu.be/MoCt9sYco10?si=XyE77deA6QFSuMYf

Rock:
-Back in Black by AC DC: https://youtu.be/wuQAg8xdmyc?si=IROrKG1avKlws1Be -
Sweet Child O Mine by Guna N Roses:
https://youtu.be/_bvTm53y_ZE?si=CInH6PmeNwSDhy5D

-Wind of Change by Scorpions: https://youtu.be/8FMpIiDo-Ic?si=0c83rEVAalKDWC_A

- The Chain by Fleetwood Mac: https://youtu.be/dGykwC0fdJ4?si=FBBK6RAPHJT4ITYo