Jeet Ajmani, Luhee Hyunkyung, Henry Windish
Integrated Music and Multimedia
7 December 2024

Arcade Game Final Project

Description

Breathinity is a virtual soundscape where your breath shapes the world around you. Each inhale and exhale generates unique sounds and visuals, creating a deeply personal and immersive experience. To accomplish this, our project features three components: a respiratory sensor, a VR headset, and generative audio.

Our respiratory sensor is built around an Adafruit rubber stretch sensor. As the sensor is stretched, its resistance increases. Using a Mega 2560 microcontroller we can measure this change. As resistance increases, "breath" increases. On the hardware side we scale this resistance reading from zero to one for each user. At zero, the sensor is unstretched. At one, the sensor is maximally stretched. These readings are collected on a laptop and then sent via OSC to the headset.

For the VR headset we borrowed a Meta Quest 2 from the gadgets department of the Georgia Tech library. Our initial project idea was to include some type of immersive virtual reality environment, and the somewhat limited selection of headsets at the library led us to using a Meta Quest 2. Unfortunately (and the source for most of our technical challenges throughout the project timeline), Meta does not provide their "Meta Link" software to MacOS computers. Even for Windows PCs, running the software requires a powerful GPU. Since running the game through the Unity application was out of the question, we found the lengthy workaround of building the game as an Android APK and sending it to the headset to run independently. This process involved jumping through even more hoops, but ended up working out in the end. It involved a full factory reset of the Quest, creating a new user profile, using two-factor authentication to activate the developer features, and connecting to the GTVisitor network on campus. In summary, the game is created using Unity on a MacOS computer and the game is shipped to the headset via a usb-c cable as an Android APK which can then be run as a standalone application on the headset.

Procedural audio is generated by Supercollider on a separate laptop. OSC messages are sent from Unity based on the movement of the objects. A real-time sound synthesis system designed with SuperCollider, to dynamically generate sounds based on the spatial position and distance of spheres. The sound system receives spatial data of spheres through OSC (Open Sound Control) messages and uses this data to create or update corresponding synthesizers in real-time, adjusting sound parameters as spheres move or are added. Sound synthesis is implemented using the SynthDef(\sphereSound) in SuperCollider, where the spatial and distance data of each sphere are mapped to sound parameters such as base frequency (baseFreq), amplitude (amp), stereo panning (pan), reverb amount (revAmt), frequency modulation (jitterAmt), and filtering. Closer spheres produce unstable, low-frequency harmonics, while distant spheres generate stable, high-frequency harmonics with enhanced reverb for spatial depth. Changes in sphere positions directly influence the texture and dynamic properties of the sound, with harmonic scaling, noise layers, low-pass filtering, and stereo spatialization contributing to an immersive auditory experience. This system is giving the interaction to multi-layered sound design in interactive sound environments.

Challenges and Changes

The majority of our challenges stemmed from creating a network of devices with the capability of altering the state of one another. If we had a capable PC, this could have theoretically been accomplished with a single computer, but utilizing a network has its own advantages such as conserving CPU/GPU and ultimately we learned about successfully developing independent components that unify to create an aggregate environment.

The extOSC library in Unity was a huge help for sending and receiving OSC messages on the Meta Quest. However, since the laptop on which the game was developed and the Quest have different IP addresses, the game needed to be modified every time the OSC message reception functionality was tested. Otherwise the headset wouldn't have an open communication channel on its IP and instead the laptop would be looking for messages. This process led to a very stressful, yet nonsensical issue where the IP for each device would randomly change once every few days and Unity wasn't able to build the game to send it to the headset with its new IP. The solution was so arbitrary that it involved building the game without the OSC functionality, then rebuilding it in its optimal state to send it again, and this solved our issue

Collaboration.

Each group member lead one of our three components. Henry lead on the sensor, Jeet lead on the 3D environment, and Luhee lead on the procedural audio and object-based interactive environment. This division allowed us to work independently on each section. We shared our progress through several meetings.

We could have improved our collaboration by prototyping the communication between our components. It was only after nearly finalizing each component that we worked to connect the system. We had a last minute issue with sending OSC messages to the VR headset and had to rework some of our game to help resolve that issue, but we could handle it before the presentation. Overall, we learned that our collaboration would have been improved by integrating those projects earlier in our timeline.