VRLC Music: Live Programming of Musical Instruments in Virtual Reality

By Eva Jain
Advisor: Professor Steve Tanimoto, Department of Computer Science at the University of Washington

Abstract:

In this project, we developed a framework or software called “VRLC Music” to allow live coders to create new musical instruments in an Augmented Reality/Virtual Reality (AR/VR) environment. This project used VRLC Alpha, a python based live coding environment for AR/VR developed at University of Washington.

Summary

Live coding is typically used in the generation of music, so that a user can create music in real time by incorporating feedback in real-time without recompiling the program. Traditionally, music is generated by live coders using a computer screen by controlling the synthesizer. Virtual Reality (VR) provides live coder with an immersive experience where the live coder or the user is immersed in a virtual, 3D world, and has an ability to generate or experience music without leaving the virtual environment. The goal of this project was to implement a facility that lets a team of two people perform a "live coding duet", where one person is sitting at a desktop or laptop computer programming in Python, and the other person is using the Oculus and coding in VR. The code of each performer would be driving music synthesis in real time, and they would be co-creating a performance that can be experienced by others over the web. To meet the project objectives, we used VRLC Alpha, a web-based programming environment for AR/VR developed at University of Washington, to develop a framework that allows live coders in VR to create various musical instruments on the fly.

“VRLC Music” Framework

Dependencies:

A-Frame:

A-Frame is an open-source web framework for building VR experiences and supports common VR headsets like the Oculus, etc. A-Frame can be used to create 3D objects called entities as well set their properties, such as shape, color, and scale. There were also many events you can set, which run events based on user input.

July-August 2022

Web Audio API:

Web Audio API uses oscillators to play sounds at specific frequencies, which can be used to play notes. You can turn the oscillator on and off to play the tone.

VRLC Alpha:

VRLC Alpha is a web-based programming environment for AR/VR developed at University of Washington. The VRLC Alpha is hosted on a University of Washington server and allows the user to live program in Virtual Reality using the built-in python editor. It consists of the following:

Python module: This module allows python code to be run in a browser and exposes all the VR functionality to the live coder. Running the built-in python module allows Virtual Reality scene to get updated with the new code, which sets up live coding. The python module uses functions from the A-Frame framework.

Python callable JavaScript module: This module allows A-Frame functionality to be accessed from the "python module." The A-Frame library is written in JavaScript and HTML and is called from python using python callable JavaScript API. For example, to add functions to the python module, such as playNote to play a note in the Virtual Reality scene, it is coded as a python callable JavaScript function in the JavaScript file and called in the python module.

"VRLC Music" Framework Details:

We used the A-Frame framework to create entities or 3D objects and connected these entities to the Web Audio API, creating a musical instrument. Using gaze-based interaction in the Oculus Quest 2 (Oculus), users look at an entity and play a sound. This was integrated with the VRLC Alpha server, where live coders use the online python editor on computer or Oculus to create a scene that shows up in the Virtual Reality environment. The customized functions meant for Virtual Reality were written in JavaScript and called in Python.

To create musical entities in the Virtual Reality scene, we developed a JavaScript function called CreateEntity, which can be customized using the following parameters for creating a playable key:

mixInObj – the type of geometry primitive (e.g., cube, sphere, etc.)

OuterObj – position or other attributes of the object

InnerObj – class~intersectable for creating an interactable entity

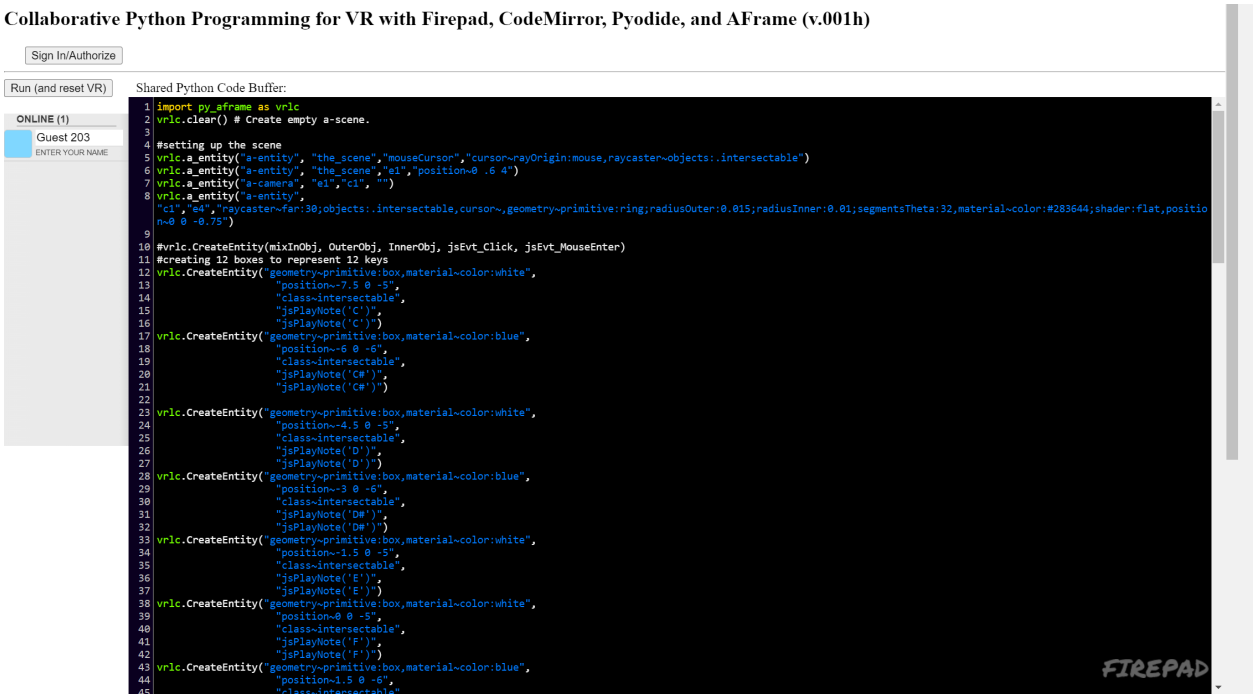
jsEvt_Click – the function that runs when the object is clicked

jsEvt_MouseEnter – the function that runs when the object is hovered on

To play different notes, we developed a JavaScript function called jsPlayNote(note) that is called on each event. To make a note, we made the oscillator run at that note's frequency. Then, using a delay, we played the note for 0.75 seconds.

Output

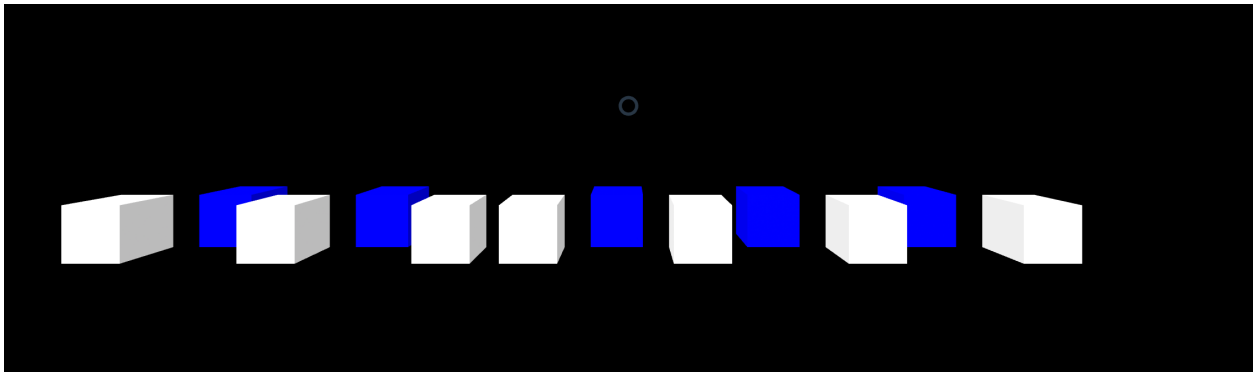
The following is the screenshot of the VRLC environment and live coders can either use a static file or commands to create new VR entities:



For example, the following code generates the output shown in Figure 1 where gazing on different boxes produces different notes.

```
vrlc.CreateEntity("geometry~primitive:box,material~color:white",
                  "position~-7.5 0 -5",
                  "class~intersectable",
                  "jsPlayNote('C')",
                  "jsPlayNote('C')")
vrlc.CreateEntity("geometry~primitive:box,material~color:blue",
                  "position~-6 0 -6",
                  "class~intersectable",
                  "jsPlayNote('C#')",
                  "jsPlayNote('C#')")

vrlc.CreateEntity("geometry~primitive:box,material~color:white",
                  "position~-4.5 0 -5",
                  "class~intersectable",
                  "jsPlayNote('D')",
                  "jsPlayNote('D')")
vrlc.CreateEntity("geometry~primitive:box,material~color:blue",
                  "position~-3 0 -6",
                  "class~intersectable",
                  "jsPlayNote('D#')",
                  "jsPlayNote('D#')")
vrlc.CreateEntity("geometry~primitive:box,material~color:white",
                  "position~-1.5 0 -5",
                  "class~intersectable",
                  "jsPlayNote('E')",
                  "jsPlayNote('E')")
vrlc.CreateEntity("geometry~primitive:box,material~color:white",
                  "position~0 0 -5",
                  "class~intersectable",
                  "jsPlayNote('F')",
                  "jsPlayNote('F')")
vrlc.CreateEntity("geometry~primitive:box,material~color:blue",
                  "position~1.5 0 -6",
                  "class~intersectable",
                  "jsPlayNote('F#')",
                  "jsPlayNote('F#')")
vrlc.CreateEntity("geometry~primitive:box,material~color:white",
                  "position~3 0 -5",
                  "class~intersectable",
                  "jsPlayNote('G')",
                  "jsPlayNote('G')")
vrlc.CreateEntity("geometry~primitive:box,material~color:blue",
                  "position~4.5 0 -6",
                  "class~intersectable",
                  "jsPlayNote('G#')",
                  "jsPlayNote('G#')")
vrlc.CreateEntity("geometry~primitive:box,material~color:white",
                  "position~6 0 -5",
                  "class~intersectable",
                  "jsPlayNote('A')",
                  "jsPlayNote('A')")
vrlc.CreateEntity("geometry~primitive:box,material~color:blue",
                  "position~7.5 0 -6",
                  "class~intersectable",
                  "jsPlayNote('A#')",
                  "jsPlayNote('A#')")
vrlc.CreateEntity("geometry~primitive:box,material~color:white",
                  "position~9 0 -5",
                  "class~intersectable",
                  "jsPlayNote('B')",
                  "jsPlayNote('B')")

FIREPAD
FIREPAD
```

Figure 1

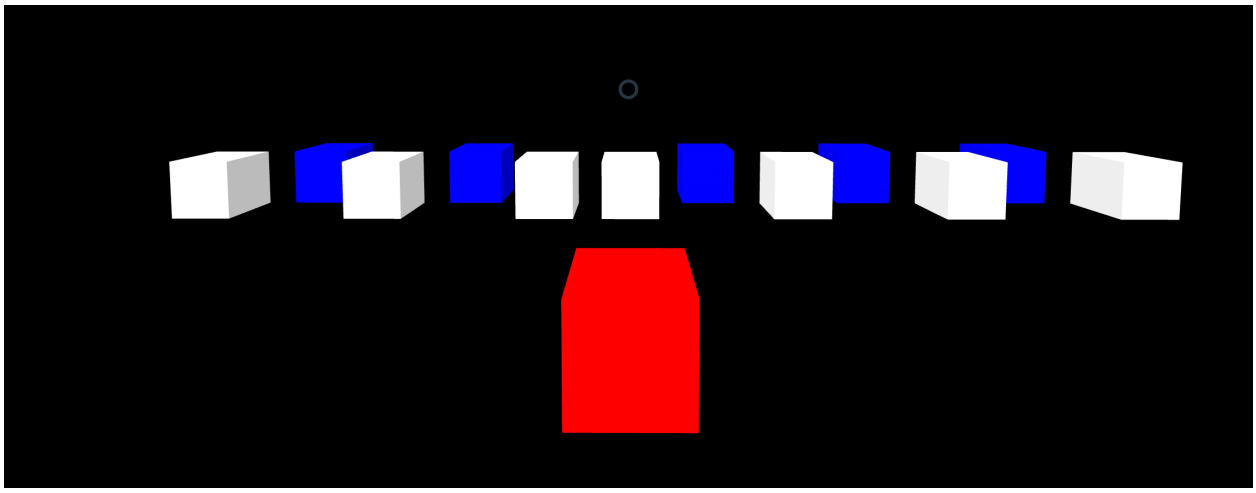


If we add another musical entity using:

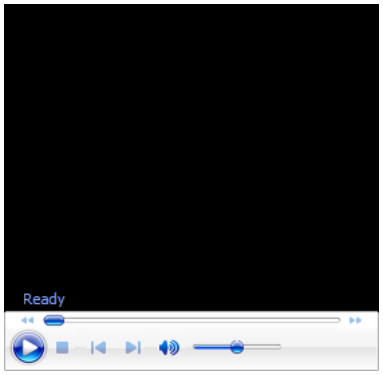
```
vrlc.CreateEntity("geometry~primitive:box,material~color:red",
                  "position~0 0 0",
                  "class~intersectable",
                  "jsPlayNote('B')",
                  "jsPlayNote('B')")

FIREPAD
```

This command will produce the output shown in Figure 2.

Figure 2



Here is a working instrument that can be played on computer or on the Oculus:



Challenges:

Testing in AR/VR Environment: The code was tested using local host as it wasn't feasible to update VRLC Alpha server because it is hosted on a University of Washington server. However, Oculus doesn't connect with local host because it doesn't have SSL certificate (or https) be default. As a result, we created a repository on GitHub so that we can test the code on the Oculus using HTTPS.

Linking Code Written in Multiple Languages: In the A-Frame library, code was written in a mix of HTML and JavaScript, but the VRLC Alpha environment is in python. So, we created python callable JavaScript functions to call the code from the VRLC Alpha Python module.

Learnings:

Debugging: Browser developer tools, especially the console, were extremely useful in figuring out what part of the program wasn't working. We used the console to see if there were any errors in the program, as well as test, print, and view properties.

JavaScript/HTML: I learned how to code in different languages (HTML, JavaScript, and Python) and how different languages interoperate, as I called various JavaScript functions from a python module.

AR/VR: I learned how to program for AR/VR and how to create a user interactive program. I learned various ways to test my code for the VR environment as you can't easily test code directly in VR.

Live Coding: I got exposed to a live coding environment where program changes are reflected in real time. This environment is extremely useful in education and improvisation.

Organization: I realized the importance of keeping the files organized so that the files and webpages are loaded correctly.

Code:

The code is hosted on GitHub at the following location:
https://github.com/eva032/eva032.github.io.

This link runs the code:
https://eva032.github.io/vrlc001i.html.

Future:

Our framework sets up a "live coding duet" where one individual programs music in Python, while the other codes in the Oculus.

References:

Lowis, C. (2013, June 5). *Playing notes with the web audio API part 1 - monophonic synthesis*. Blog RSS 20. Retrieved August 18, 2022, from https://blog.chrislowis.co.uk/2013/06/05/playing-notes-web-audio-api.html

Ngo, Marcos, & McCurdy. *A-Frame*. Retrieved August 18, 2022, from https://aframe.io

Tanimoto, S. (2022). *VRLC Alpha – Collaborative Python Programming for VR with Firepad, CodeMirror, Pyodide, and AFrame (v.001h)*. Blog RSS 20. Retrieved August 18, 2022, from https://lamprovles.cs.washington.edu:8443/vrlc001i.html