Computational Sound Final Project

Federick Gonzalez, Bryanna Geiger, Yael Cohen

October 23, 2020

Abstract

We are using WebAudio in JavaScript to make a variety of synths in order to simulate actual orchestral instruments. These synths are then linked to a keyboard to emulate a digital keyboard where the user can select which instrument is being played on the keyboard. After incorporating these various synths and keyboard functionality, we applied an additional feature that has some of our synth instruments playing a segment of Federick's arrangement of a medley from the movie "Avengers Endgame." Lastly, we also implemented a Markov Chain model to each synth playing in our Avengers arrangement so that they are able to have all synths playing and play new music based on their given note progressions simultaneously. A demo of the web application can be found right here , and the code for this project and related libraries can be found in the references section.

1 Introduction

1.1 What is WebAudio?

WebAudio is an API that allows the user handle audio operations. For instance, the user can work with audio nodes, which function as oscillators that can have varying wave types. These audio nodes can be linked together in different ways to provide a variable sound output. We can adjust the intensity of the different sounds or add in different effects such as low-pass filters, biquad filters, white noise, reverb, and so on.

1.2 Synths:

Using WebAudio, we are able to build our own synths by applying techniques in WebAudio. We have made a variety of different synths thus far. Our current functionality supports high & low brass, high & low strings, high & low winds, and simple percussion synthesizers.

Each instrument is handcrafted from a supply of oscillators and filters applied to said oscillators. This allows us to use multiple voices in an attempt to match the timbre of the instrument. We have utilized techniques in WebAudio

such as through the use of various filters, wave forms, and envelopes to make these frequencies sound as close to the actual instruments as much as possible.

1.3 Sound Generation

We used a series of oscillators, envelopes, and filters to generate sounds to emulate actual instruments.

An ADSR envelope allows us to adjust the attack, decay, sustain, and release of a sound. By experimenting with these, we are able to generate a wider variety of sounds. For instance, you can generate a more staccato sound (similar to synthesizing some percussive sounds) by having a short attack time and a faster release time.

We also used various filters. Each filter allows us to adjust different aspects of the sounds being generated. For instance, using a low-pass filter makes the sound more muffled. Using the low-pass filter for the Trombone sound allowed us to create some nice low warm frequencies for our brass section that we would not have been able to generate as closely without the use of a filter.

1.4 Overview

Using our handcrafted synthesizers, our project implements a digital keyboard, allowing the user to select one synthesizer and play it on their own keyboard. Our project also plays a segment of an arrangement of Avengers done by Federick last spring, and lastly it also implements a Markov Chain model which is applied to each synth to generate new music based off of the parts for Avengers.

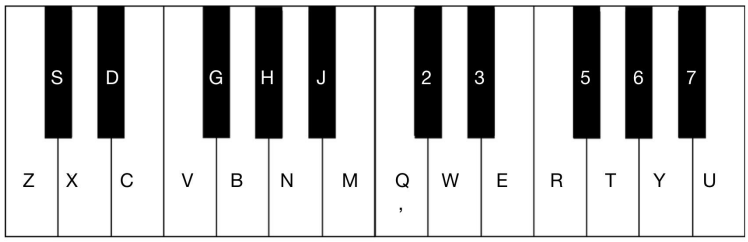

Figure 1: The laptop to Virtual Keyboard configuration

1.5 What are Markov Chains?

Markov Chains are a sequence of discrete random variables in a model, where the next state depends on the previous state. Essentially, Markov Chains are a set of transitions and these transitions are determined by a probability distribution which is determined purely by the previous state. This means that Markov Chains do not take into account the entirety of all the preceding states, but just the one directly prior to it. Simply, Markov Chains are models that describe a

sequence of possible events. While they are often used in terms of predicting word transitions, they can be used in music as well! This implementation includes giving music as an input; we chose Federick's arrangement of Avengers because it was supposed to be performed last spring by Columbia Pops Orchestra but Covid obviously put things on hold. Thus, we thought it would be a fun idea to bring Federick's arrangement to life by having it be played by our synths as well as having Markov Chain functionality attached to it.

2 Implementation & Design Choices

2.1 Implementation

Our project consists of a variety of synthesizers (Brass, Strings, Woodwinds, Percussion). We are utilizing the Tuna.js library to apply different filters to our handcrafted oscillators in order to emulate each instrument's unique sound.

The brass instruments, for example are created using 5 oscillator notes, initially set to the same frequency. Each oscillator is then set to a sawtooth wave and detuned slightly, by no more than 10 cents. These oscillators are then connected to a gain node with an envelope that was experimented with to simulate the sound of air moving through tubing. These are then passed through a lowpass filter, which has an oscillating cutoff frequency using an LFO, or a low frequency oscillator. This sound then has a moog filter placed on it, which alters the sound based on octaves above a given frequency value, and then finally a convolver was added to give the brass section a reverberating sound, just like the instruments themselves. The string and wind instruments followed similar constructions, but with different combinations of filters, oscillators, and detune values. The moog filter and the convolver both come from the Tuna.js library, along with the overdrive filter used in the strings and a few other filters that were created but unused. These filters are pseudo-Audio nodes, as in they pretend to be gain nodes, but must be connected to either a gain node or an audio context.

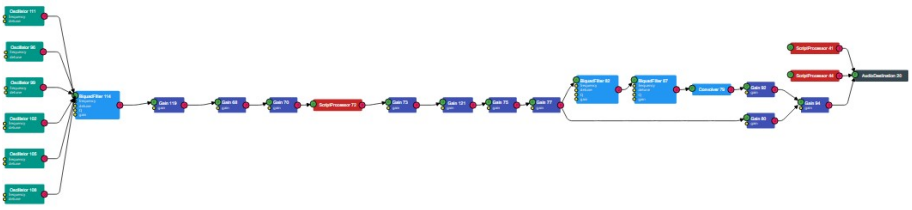

Figure 2: This is the WebAudio map of the Low Brass synth. Several of the gainNodes are actually Tuna.js filters

The next aspect of implementation involves playing Avengers with our synths. We translated the sheet music to an array of dictionaries to inform which notes were to be played when and for how long. Next, we created a Player parent class with Brass, Wind, and String subclasses to create objects for each instrument.

Within each subclass, we created a method to prep the instrument by instantiating, connecting, and starting oscillators, lfos, filters, in the same configuration that was done for the general keyboard. This was done in order to keep the keyboard functionality for a typical user, and in addition to having an option to play the Markov Chain creation. Then, we included a method that would actually assign specific frequencies and gains to the oscillators and filters created in prep, and by also designing sound envelopes and filter envelopes. The notes would then be played by being read through in a for loop. The information stored in each of these lists would be the instrument type, the note frequency (in midi), the start time for each note, and the end time for each note.

In terms of applying Markov Chain models to our various synths, a key function here is genNGram, which starts off by creating an empty object. The function then reads through the characters in the String of notes. It then looks at the ngram object and sees if it's a new one, and if so it makes an empty array, and then adds the next character as a possible outcome. We also made the user have control over the order "n" of the Markov Chain. The end goal was to then generate a sequence of notes based off of these Markov Chains.

The star of the show, however, is the function genMarkNotes, which reads in the music and adds it into a String called text. Next, it generates an ngram! Then, starting with an arbitrary ngram, it generates new characters and picks one randomly that will get played thanks to the function Choice that takes into account the probabilities of the Markov Chains generated. This new generated note is then pushed to outNoteList to get played.

2.2 User Interface

The interface itself consists of a few different features: 1) A drop-down menu of synths and 2) A play Markov Chain button with a slider.

The drop-down menu allows the user to select an individual synth. Once that synth is selected, the user can then use their keyboard to generate sound, essentially emulating a digital keyboard with various synths. This feature allows for a greater amount of user engagement.

The Play Markov Chain button adds an additional feature to this project. By pressing this button, a short automated composition will be played. The composition played involves playing all of our various synths simultaneously. There are two aspects to this component. The first is that the original Avengers composition using our various handcrafted synths will be played. This implementation purely uses the handcrafted synths being played simultaneously to generate this short composition.

The second component to the Play Markov Chain button is the actual addition of Markov Chains. The Markov Chain audio generation is played directly after the original Avengers composition. Additionally, the sound being generated is dependent on the implementation of Markov Chains which are being applied to each synth. The N value, ranging from 1 to 12, for the Markov Chain can be set by the user by using the slider to the right of the buttons. We felt as if the slider added an additional engagement component for the user to interact

with this interface. Additionally, the frequencies being played are being output to the console for reference.

We felt that the drop-down menu and play button with a slider for Markov Chain functionality incorporated all of the necessary functionality of our project given the automated composition as well as the user engagement component with a digital keyboard utilizing our handcrafted synths. In terms of the visual aspect, the changing gradient of the background added a more aesthetic component rather than having a static background with just a drop-down menu and play button. Additionally, as our project focuses on computational sound and with our focus on emulating various orchestral implements, adding in a music note cursor, while not impacting the functionality, seemed to be more of a fun touch for the user.

3 Results & Analysis

3.1 How did our project turn out?

At the beginning of this project, the webaudio-instruments library (which can be found here) was used as a basis for figuring out how to convert different instruments into synthesizers. Due to a lack of clear documentation and a sheer abundance of code to understand, we listened to the demo, at the hyperlink above, and decided the main goal was to create synths that sound more realistic, if not perfect.

Ultimately, we were able to create a variety of synths using WebAudio to emulate actual orchestral instrument. While these sounds might not be the same as the actual instruments, we were able to explore different techniques to simulate the sounds as close as we felt we were able to, and are impressed but how much JavaScript is able to create through the WebAudio API (and of course, our programming skills). The array of synths we created allowed us to play a short composition which was partially pre-coded and partially computer generated, and the finished product of this combination sounds as if it could be its own piece of proper orchestral music, containing 6 alternating voice lines which are fully in harmony.

Being that users are able to select individual synths and play them through the use of their own keyboards, they could record individual sounds and overlay them in order to make their composition more personalized, or if they just want to get a sense for how a certain measure or part could sound for a given instrument.

The Markov Chain Models were an interesting application to this project, allowing us to generate an automated composition. While playing various synths at one time is not currently possible with the setup of our web application, we were able to apply Markov Chain Models to all of our synths and ultimately have them be played simultaneously.

3.2 Potential Applications and Extensions

In addition to our final result, there are various other applications and potential extensions to this project. Going forward, we could incorporate different aspects to add to the functionality and user engagement. For instance, we could incorporate pitch set theory into the automated composition, allowing different transformations to be performed on either the original input or the generated Markov Chain composition which would still sound agreeable to the rest of the piece. Another potential extension could be to explore the idea of being able to read in a midi file that would allocate parts appropriately and play back the polyphonic music that we have composed, or at the very least read input from a Midi device and be able to play that.

One potential extension could be to add some sort of visualization for the notes being played - while this would not impact functionality, it would add to the user engagement aspect of the interface and would be interesting to explore. Currently we have a changing gradient background which is intended to give off a soothing, chill vibe to the user as they relax and play some music, however this gradient could change color based on the notes being played or, at some point, the chord being played, minor keys displaying cooler colors and major keys displaying warmer colors. Additionally, while we do currently support a variety of synths, it would be interesting to add additional synths to make a wider array of instrument sounds. At the moment, the low and high variations of different instruments is simply a change in frequency, and the synths act as whole sections rather than just individual instruments. This concept would allow us to delve deeper into the sounds and compositions we could make with this project.

4 Discussion

From this project, there are definitely things to consider going forwards. For instance, it would be easier to debug code by breaking the program into different components. Working with smaller files instead of a few large ones spanning almost 1,000 lines of code would make it easier to troubleshoot problems that arise with the program and test smaller aspects of the code. Additionally, in terms of readability and for easier debugging, it would also be worth looking at refactoring components of the code. For instance, each of the separate instruments could be contained in separate js files, or perhaps the user interaction could be one file, instrument creation in another, and musical functionality in a third.

Additionally, one aspect to consider is the limits of WebAudio. While we probably did not reach the upper limits of what WebAudio is capable of handling especially in terms of playing different synthesizers at once, the original structure of the code seemed to near the boundaries. It's important to keep in mind not just the WebAudio API's capabilities, but also the machine capabilities, and work on generating the most efficient and therefore effective code possible.

Another potential reason for having the initial difficulties with our audio playing could have been due to instances of clipping which resulted in the audio almost sounding as if it was being crushed, instead of being able to hear the individual notes being played. This was mainly fixed by limiting the amount of instruments that could be played at one time from hundreds down to 7, and reworking the logic of the program so that each instrument is only created once, and therefore there is no possibility to create infinite oscillators.

Furthermore, we were having difficulties with only being able to play the audio one time through, meaning no audio would be played the second time through when we clicked on the play button. This was remedied by including the time of the button press in the creation of each individual note. Each note initially had a start time and an end time, and the program was given instructions to start at the start time and end at the end time. However, this carried a few difficulties. The first being notes would start playing before the entire computation process was complete. The second being that the times were based off of the initial compilation of the web page, allowing the Markov Chain generation to only be heard once. This was remedied by adding an offset of 1 second, to allow for computation, and adding the time of the button click to this offset. When this was added to the start and end time of each note, then the piece could consistently be heard after each button press.

Overall, the group was able to produce a fully functioning web application that consists of a digital keyboard with 9 different synthesizer options, along with a Markov Chain composition extender which can play multiple synthesizers at once.

References

[1] Tuna.js library

[2] Computational Sound Final Project code