Synthesizer Software

Contents

1. Abstract
Do this at the end

2. Analysis

2.1. Client
The client is my friend, Marli Carroll.

2.2. Questions & Prompts
1 Discovering Problems
  1.1 "What problems have you encountered using online technology when creating music"
  1.2 "What are some features that you use frequently?"
  1.3 "How do you get around these problems right now?"
2 Project Scoping
  2.1 "Will this project be rolled out immediately or gradually into your music?"
  2.2 "Would this software be used for commercial use or only for personal use?"
  2.3 "Could you describe any explicit requirements for the software at this stage, and if so could you separate them as potentials and definite?"

2.3. Notes From Interview
1 The client describes their problems to be intertwined with having issues with not having ease of use when creating synthetic sounds on software
2 Regularly used features that includes modulating the frequency and pitch of sine waves on software such as FL Studio.
3 Enjoys using sliders and wheels regularly found on midi keyboards
4 Client describes having something separate from google, therefore not an extension
5 Customizable buttons and sliders would be useful because he likes to change out key binds
6 All requirements are featured below.

2.4. Conclusion From Interview
1 Functionality
  1.1 Will be used to prototype wave sounds with just one application
  1.2 Waves should be manipulated on this software
  1.3 Manipulation of waves will include changing the frequency, pitch, (ADD TO THIS LATER)
  1.4 Application must produce various sounds
  1.5 Sliders must be incorporated and be sliders for changing volume or amplitude.
2 Visuals
  2.1 Intuitive: All sine wave buttons should be next to each other
  2.2 The slider(s) should be smooth when moving up and down for a better GUI
3 Technologies
  3.1 Files must be .WAV to be compatible with Digital Audio Workspaces such as Logic and FL Studio

2.5. Research

2.5.1. Wavetables

Plotting a wave using data points onto a table with an x-axis of time and a y-axis of vertical displacement, there can be an accurate wave that can be used to integrate into code.

2.5.2. Wave Components

1 Gain
- The ratio of input to output voltage (Ω). A scale factor of the gain can stretch the amplitudes of waves.

2 Temporal Frequency
- How often a wave repeats itself (one oscillation). An increase in oscillations can increase the pitch of a sound, relative to keys on the piano as you get scale up to the top of the piano.

3 Volume
- How loud the output is. Volume does not affect the input, unlike gain despite both increasing amplitude.

2.6. Technologies

1 Platforms
- Must be compatible with Windows 10 & 11

2 .Net Audio Library
- A C# audio library for audio playback. This will be used for early testing stages for algorithmic code.

3 React
- JavaScript web development library to construct a local GUI.

4 Tailwind CSS
- Front-end HTML framework to develop a modern-looking GUI with component blocks.

2.7. Algorithms

2.7.1. Karplus Strong Algorithm

This algorithm will simulate string instruments using data points in a wavetable.

2.7.1.1. Periodic decay of a plucked string [10, p2]

Simulating a periodic vibration of a string is quite simple when p = pitch, given the equation:

$$p + \frac{1}{2}\text{samples}\left(\text{frequency }\frac{f_s}{p + \frac{1}{2}}\right)$$

Unfortunately, real string vibrations on a string instrument aren't perfectly periodic, changing with time. This means there is a level of randomness to the decay of strings that needs to be simulated to get a more realistic tone. The level of randomness changes depending on what string instrument is being simulated.

2.7.1.2. Simulating Randomness [9]

When plotting a wavetable with the Y axis as the amplitude in the range: $x \in (-1, 1)$ and the X axis as time(T), Each point that makes up a wave can be under the Karplus Strong equation that can simulate randomness through manipulating probability:

$y(x,t) = \frac{1}{2}(s_o(x,ct) - s_o(-x+ct)) + \frac{1}{2c}\int_{x-ct}^{x+ct} v_0(u)du$

The initial position is $y(x,t)$ of the points plotted in a wavetable described above at $x + ct$ and $x - ct$ with the initial velocity over the points, where $c = \sqrt{T^\rho}$, ρ = Density.

2.7.2. Sine wave's position & time:

Direction = Right
$y(x,t) = A\sin(kx - wt + \varphi)$
Direction = Left
$y(x,t) = A\sin(kx + wt + \varphi)$

2.8. Triangle wave

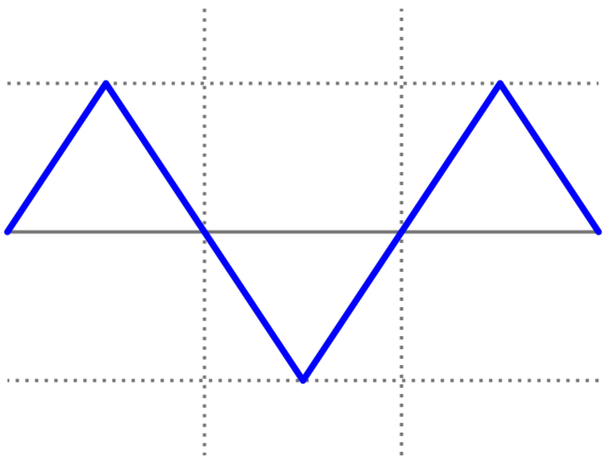


Figure 1: Triangle Wave shape [12]

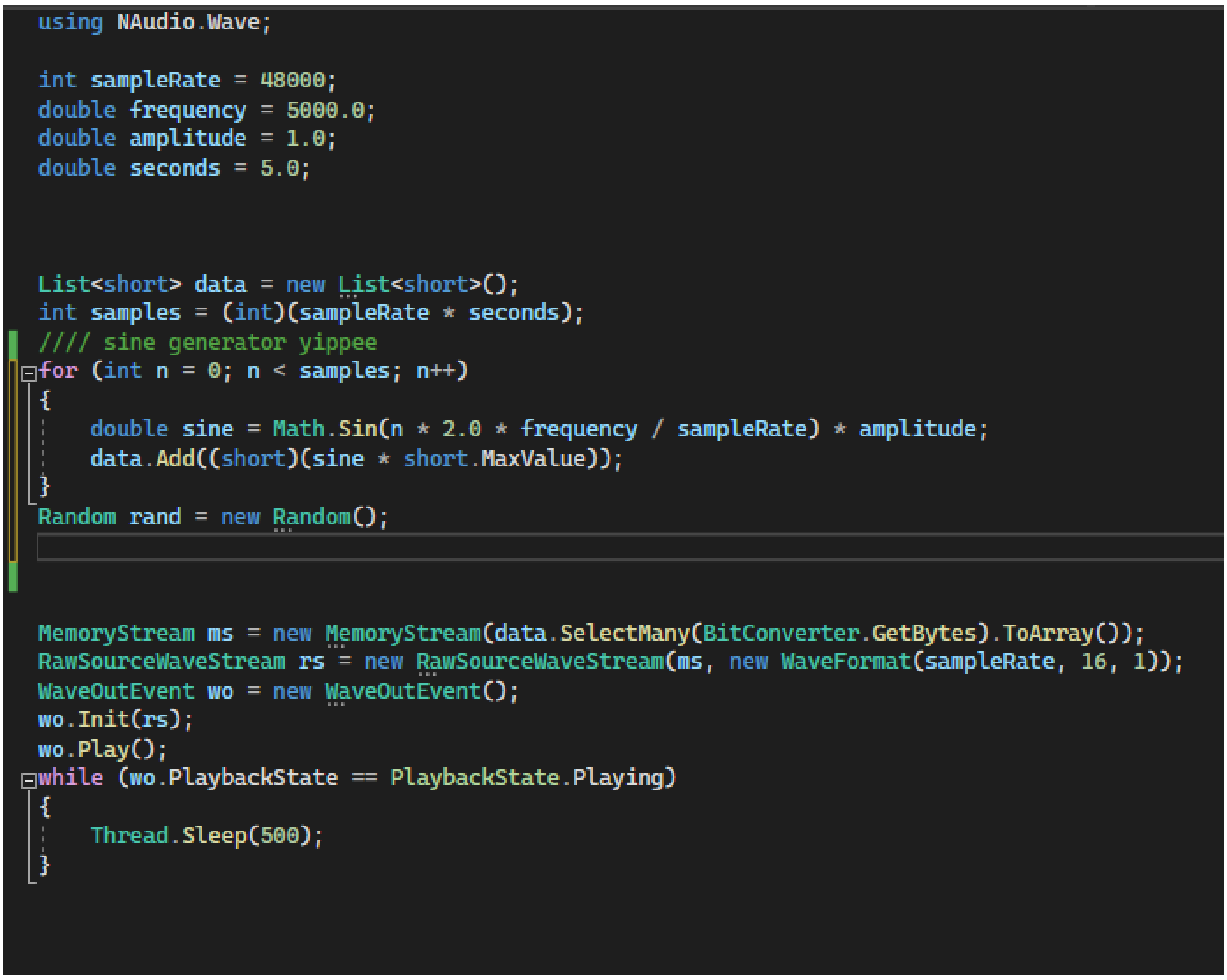
The form of the triangle wave means that there is a gradient of x which changes to negative at the peak of the wave, and changes back to positive at the trough within the range [−1,1].

2.9. Sawtooth wave

2.10. Square wave

2.11. Prototyping

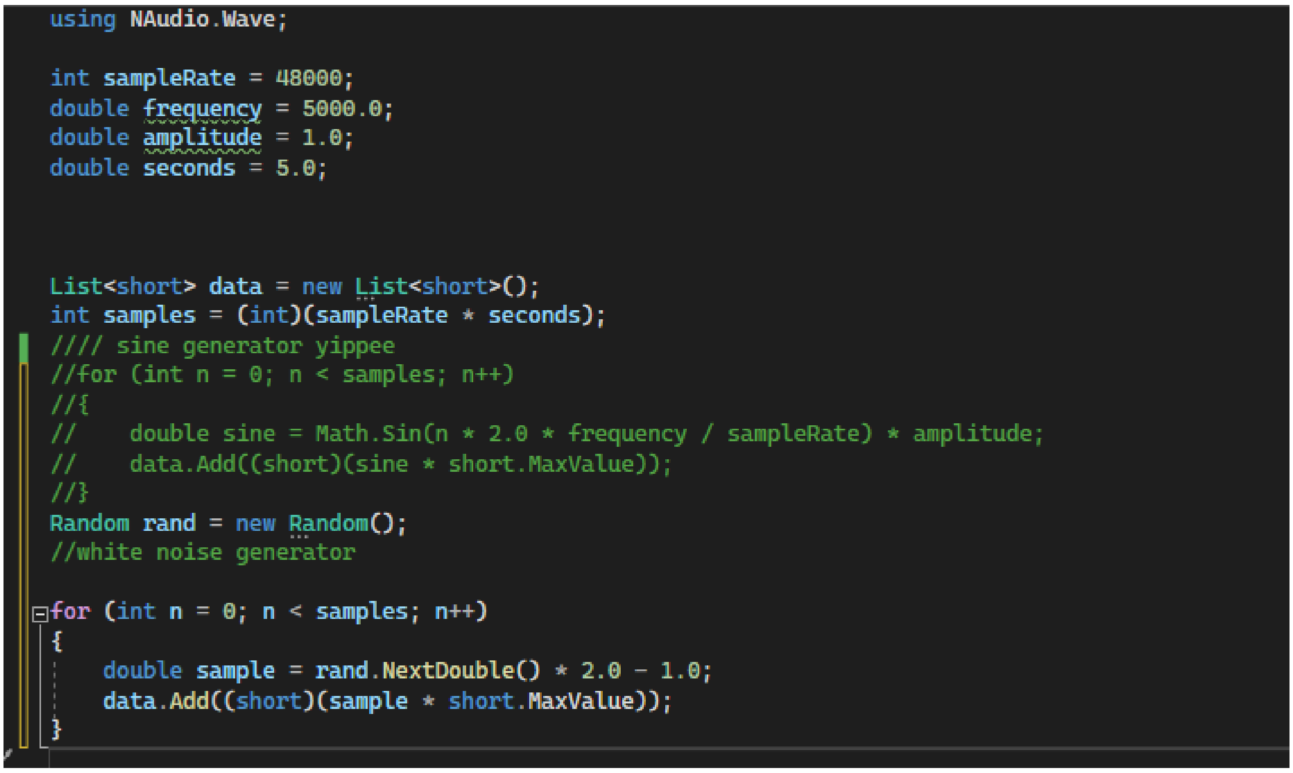
I created a prototype of audio of a sine wave to check for functionality of hardware output and to make any adjustments to using .NET audio library[6].



```
using NAudio.Wave;

int sampleRate = 48000;
double frequency = 5000.0;
double amplitude = 1.0;
double seconds = 5.0;



List<short> data = new List<short>();
int samples = (int)(sampleRate * seconds);
//// sine generator yippee
for (int n = 0; n < samples; n++)
{
    double sine = Math.Sin(n * 2.0 * frequency / sampleRate) * amplitude;
    data.Add((short)(sine * short.MaxValue));
}
Random rand = new Random();



MemoryStream ms = new MemoryStream(data.SelectMany(BitConverter.GetBytes).ToArray());
RawSourceWaveStream rs = new RawSourceWaveStream(ms, new WaveFormat(sampleRate, 16, 1));
WaveOutEvent wo = new WaveOutEvent();
wo.Init(rs);
wo.Play();
while (wo.PlaybackState == PlaybackState.Playing)
{
    Thread.Sleep(500);
}
```

Footage of the sine wave generator: https://youtu.be/qTc5Twk59sI

I also created a white noise generator to test if using any randomness in my code would effect the sound in any strange ways.



```
using NAudio.Wave;

int sampleRate = 48000;
double frequency = 5000.0;
double amplitude = 1.0;
double seconds = 5.0;



List<short> data = new List<short>();
int samples = (int)(sampleRate * seconds);
//// sine generator yippee
//for (int n = 0; n < samples; n++)
//{
//    double sine = Math.Sin(n * 2.0 * frequency / sampleRate) * amplitude;
//    data.Add((short)(sine * short.MaxValue));
//}
Random rand = new Random();
//white noise generator

for (int n = 0; n < samples; n++)
{
    double sample = rand.NextDouble() * 2.0 - 1.0;
    data.Add((short)(sample * short.MaxValue));
}
```

Footage of the white noise generator working: https://youtu.be/_v-8ep74nXs

2.12. GUI (unfinished)

2.12.1. Slider

Using the track bar class in WinForms, values can be adjusted visually and dynamically through a smooth sliding bar to change parameters of a wave. The sliders should be grouped together at the bottom left of the GUI.

2.12.2. Toggle Buttons

To implement a button that switches between two modes, I can use a check box with the appearance of a button. This can be paired with a text box that changes depending if the checkbox is true or false to indicate which mode is being used at the time.

2.12.2.1. Highlighted Buttons

2.12.3. Audio Status displayer (Oscilloscope)[11]

Stroke modules

Linear implementation

2.12.4. Wave buttons

- Each button represents each type of wave and calls a function when pressed. They do not need to be toggle buttons because each function should override the previous function and how it affects the data in a list.
- buttons should be grouped and reside next to each other with equal padding and placed to the right of the GUI

3. Project

This client requires versatile GUI software that can interact with waves and produce sounds that can be sampled. The application should simplify all basic controls for a pc into one application e.g. frequency, pitch, and the type of wave.

I will be coding in C# throughout the project.

3.1. Project Overview

The project will be broken down into: Audio Generation, Audio Manipulation, GUI Construction, and Audio Implementation. Audio Generation is the prerequisite to producing string-like audio, thus most of this stage will be understanding and using .Net Audio Library[6] to produce a wave sound. I believe Audio Manipulation will take the majority of the time during this project, translating algorithms, such as Karplus Strong theorem[9],[10], into functions with parameters that can change in value through the GUI.

3.2. Project Objectives

3.2.1. Audio

1 Prerequisite setup
  - Install/configure the .NET audio library
  - Verify functionality for audio playback

2 Audio playback testing
  - Write code for a basic wave sound
  - Check for audio lag on all hardware

3 Hardware compatibility
  - Check functionality for speakers and headphones
  - Check sound output still works when changing output devices while visual studio is open

4 Waveform generation
  - 4.1 Produce audio of:
    - Sine wave
    - White noise (to test .Random)
    - Triangle wave
      - Linear gradient throughout the noise
    - Sawtooth wave
    - Square wave
      - Can include adjustable duty cycling for testing purposes.
  - 4.2 Check for any pops or clicks during playback

5 Control over basic parameters (in code):
  - Frequency
  - Amplitude

6 Audio Manipulation
  - Test if using .Random will effect the audio quality undesirably

7 Sounds can be exported as a .WAV file

3.2.2. Karplus-Strong algorithm: advanced audio features

1 Define all constants for the algorithm
  - sampleRate
  - frequency
  - amplitude
  - seconds

2 Store data
  - Use a list called data
    - This will hold audio samples as integers (being a short) so it is mutable
  - Calculate total number of samples

3 Use a .Random variable to simulate random decay

4 Create a buffer using a queue (stack)
  - Buffer should be filled with random values from the .Random variable
  - Buffer Size = sampleRate / Frequency

5 Dequeue the top value, peek the next value

6 Sample = Dampening factor* 0.5 * (top value, next value)

7 Create a memory stream to convert the list into bytes

8 Create a audio stream to wrap the memory stream so it can be played with the correct:
- sampleRate
- 16 bit sample
- channel number (defaulted to 1)

9 Use WaveOutEvent from .Net Audio Library[6] to play the audio

3.2.3. GUI

9.1 Intuitive layout
- Distinct buttons are in convenient places
- Live view of the sound being played
- Smooth slider for turning up volumes of different pieces of software
- Use a dock panel to organize all controls and functions
- Related controls should be in groups in distinct areas

10 Waveform visualizer
- Output signals should be displayed similar to an oscilloscope
- Output visualization should be real-time (find a library for this)

10.1 Buttons

10.1.1 waves buttons that include:
- Triangle wave
- Sawtooth wave
- Square wave
- Sine wave

10.1.2 Buttons should be highlighted when clicked to show the active waveform currently used.

10.1.3 A button to toggle the Karplus-Strong decay effect
- This button should change colour when active

10.1.4 GUI should dynamically update with no stutters

11 Unique sliders

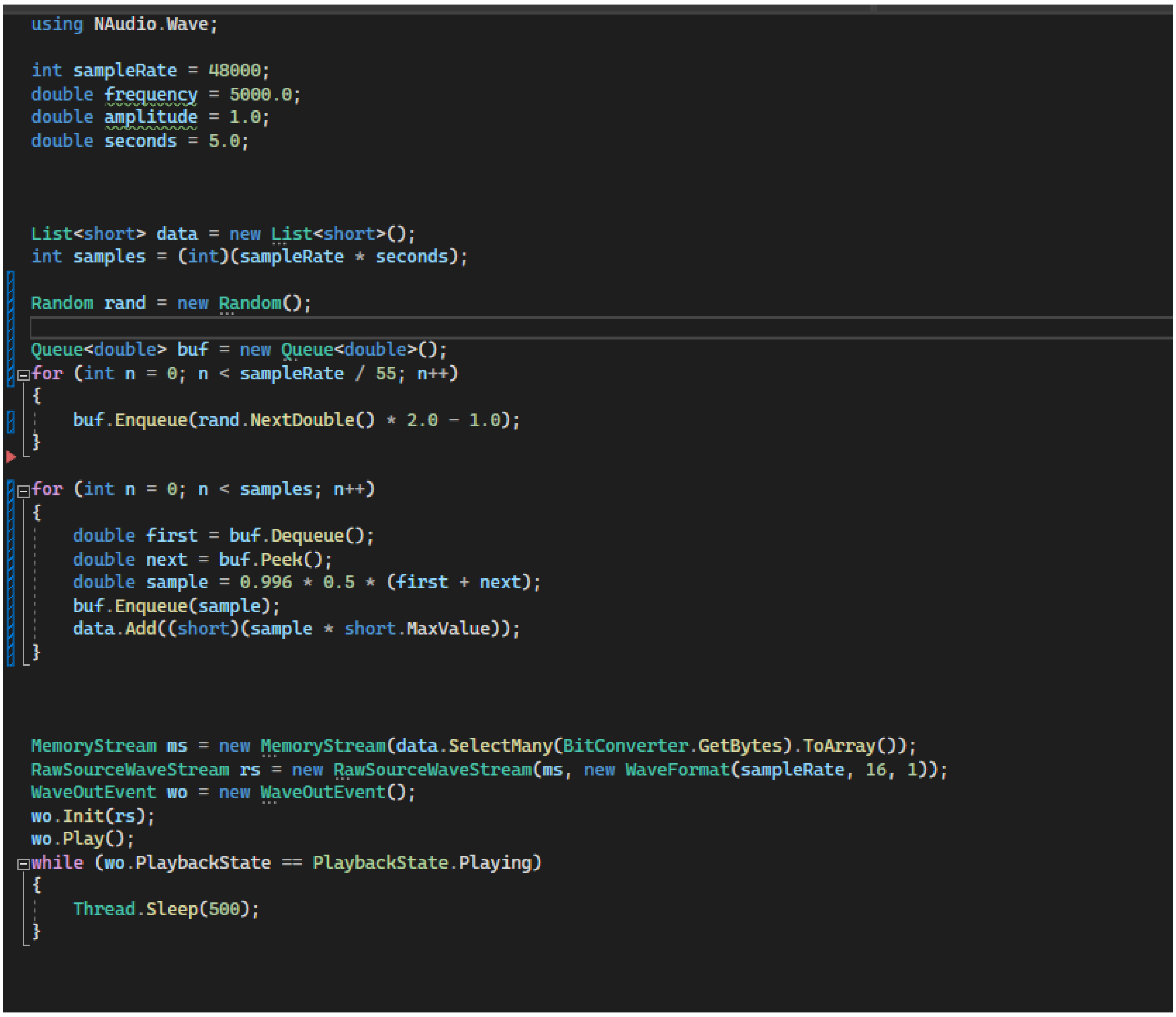
11.1 Amplitude slider
- Volume of sound adjustment
- Display current amplitude value in real time in percentage
- Sliders can be used in real time without audio stutters
- Units will be in Decibels

11.2 Frequency slider
- Scale logarithmically to have a wider frequency range (To a max of 20kHz, the sound of human hearing)

11.3 All sliders should be grouped for an intuitive design

12 Audio Status display
- At the top of the GUI, display if the audio is playing or paused

3.3. Karplus Strong Generator [9,10]



```csharp
using NAudio.Wave;

int sampleRate = 48000;
double frequency = 5000.0;
double amplitude = 1.0;
double seconds = 5.0;



List<short> data = new List<short>();
int samples = (int)(sampleRate * seconds);

Random rand = new Random();

Queue<double> buf = new Queue<double>();
for (int n = 0; n < sampleRate / 55; n++)
{
    buf.Enqueue(rand.NextDouble() * 2.0 - 1.0);
}

for (int n = 0; n < samples; n++)
{
    double first = buf.Dequeue();
    double next = buf.Peek();
    double sample = 0.996 * 0.5 * (first + next);
    buf.Enqueue(sample);
    data.Add((short)(sample * short.MaxValue));
}



MemoryStream ms = new MemoryStream(data.SelectMany(BitConverter.GetBytes).ToArray());
RawSourceWaveStream rs = new RawSourceWaveStream(ms, new WaveFormat(sampleRate, 16, 1));
WaveOutEvent wo = new WaveOutEvent();
wo.Init(rs);
wo.Play();
while (wo.PlaybackState == PlaybackState.Playing)
{
    Thread.Sleep(500);
}
```

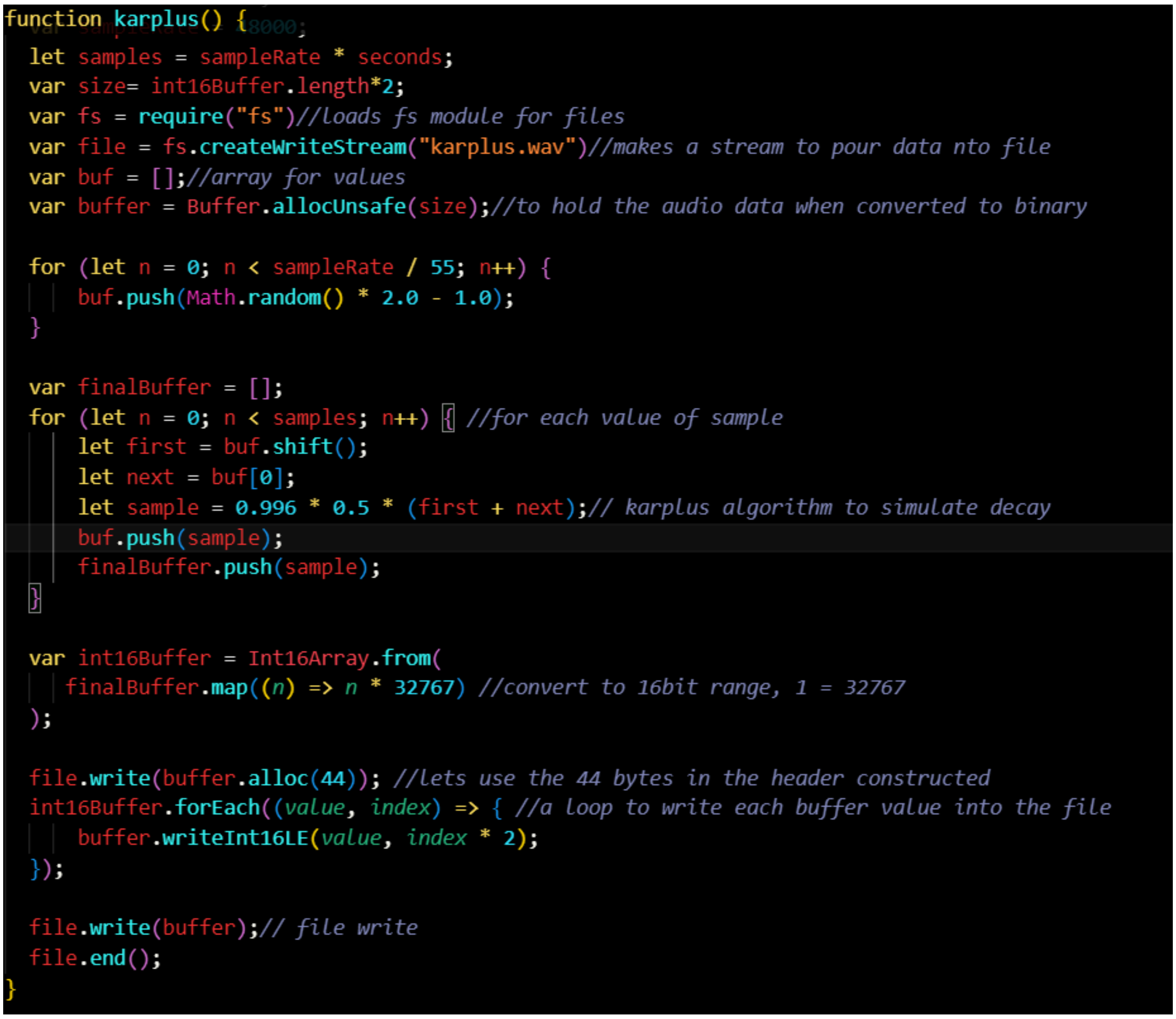
Footage of the Karplus Strong Generator working: https://youtu.be/oyP7xDqaeEk

This code is me implementing the of using randomness for decay (described in more detail at 2.7.1.2) through the use of indoctrinating the formula loosely by converting the sample data into integers, going through the function to achieve random decay, and outputting the audio by converting the integers into bytes.

By using a stack, I can go through each sample and buffer the audio by multiplying the samples by a dampening factor (in this instance the integer 0.996). After getting a new list of data, I implemented the data into a memory stream so it can be wrapped with sampleRate, bit rate, and amount of channels in a new stream called the RawSourceWaveStream. This converts all the data into bytes for the WaveOutEvent to process and output the audio.

The while loop has been used while testing to play for 5 seconds.

3.4. JavaScript Karplus Strong translation



```
function karplus() {
  let samples = sampleRate * seconds;
  var size= int16Buffer.length*2;
  var fs = require("fs")//loads fs module for files
  var file = fs.createWriteStream("karplus.wav")//makes a stream to pour data nto file
  var buf = [];//array for values
  var buffer = Buffer.allocUnsafe(size);//to hold the audio data when converted to binary

  for (let n = 0; n < sampleRate / 55; n++) {
      buf.push(Math.random() * 2.0 - 1.0);
  }

  var finalBuffer = [];
  for (let n = 0; n < samples; n++) { //for each value of sample
      let first = buf.shift();
      let next = buf[0];
      let sample = 0.996 * 0.5 * (first + next);// karplus algorithm to simulate decay
      buf.push(sample);
      finalBuffer.push(sample);
  }

  var int16Buffer = Int16Array.from(
     finalBuffer.map((n) => n * 32767) //convert to 16bit range, 1 = 32767
  );

  file.write(buffer.alloc(44)); //lets use the 44 bytes in the header constructed
  int16Buffer.forEach((value, index) => { //a loop to write each buffer value into the file
      buffer.writeInt16LE(value, index * 2);
  });

  file.write(buffer);// file write
  file.end();
}
```

I translated my back-end code into javascript to ensure compatibility with the GUI. This is so that there are no c# library issues that may have occurred if I decided to use data outputted from c# into my webUI. Given that web-dev is the easiest way to create a GUI, javascript as my language will be the most optimal because of the vast amount of tools the language provides.

3.5. GUI Construction

ADD IMAGE HERE

3.5.1. Button Implementation

I began creating buttons using React[17] and Tailwind[16] for the various waves I'd later implement with onClick functions. I added basic highlighting animations to the buttons when hovered over to show when the user can click the button.

3.5.2. Slider Implementation

4. Bibliography

[1] Wikipedia, “Sine Wave Formulas.” Accessed: Sep. 08, 2024. [Online]. Available: https://en.wikipedia.org/wiki/Sine_wave

[2] Fl studio, “Groove Machine Synthesizer.” Accessed: Sep. 08, 2024. [Online]. Available: https://www.image-line.com/fl-studio-learning/fl-studio-online-manual/html/plugins/GMS.htm

[3] Sam Arron, “Sonic Pi Engine.” Accessed: Sep. 09, 2024. [Online]. Available: https://sonic-pi.net/

[4] Peta Sittek, “Volume Master.” Accessed: Oct. 14, 2024. [Online]. Available: https://www.petasittek.com/volume-master/

[5] Asus, “Asus Dial.” Accessed: Oct. 16, 2024. [Online]. Available: https://www.asus.com/proart/software-solutions/asus-dial-and-control-panel/

[6] Naudio, “NAudio Git repo.” Accessed: Oct. 22, 2024. [Online]. Available: https://github.com/naudio/NAudio

[7] Girth Summit (latest edit), “Gain(electronics) Wiki.” Accessed: Nov. 13, 2024. [Online]. Available: https://en.wikipedia.org/wiki/Gain_(electronics)

[8] Mikhail Rynazanov, “Triangle Sine Wave Wiki.” Accessed: Nov. 13, 2024. [Online]. Available: https://en.wikipedia.org/wiki/Triangle_wave

[9] N/A, “Drexel-Karplus Strong Therom.” Accessed: Nov. 27, 2024. [Online]. Available: https://www.math.drexel.edu/~dp399/musicmath/Karplus-Strong.html

[10] A. S. Kevin Karplus, “Digital Synthesis of Plucked-String and Drum Timbres.” Accessed: Nov. 27, 2024. [Online]. Available: https://www.jstor.org/stable/3680062?seq=2

[11] Beto-Rodriguez, “LiveCharts GitHub Docs.” Accessed: Dec. 06, 2024. [Online]. Available: https://github.com/Live-Charts/Live-Charts

[12] Omegatron, “TriangleWaveImage.” Accessed: Dec. 06, 2024. [Online]. Available: https://en.m.wikipedia.org/wiki/File:Triangle_wave.svg

[13] IconDuck, “sawtoothIcon.” Accessed: Jan. 23, 2025. [Online]. Available: https://iconduck.com/icons/91203/sawtooth-wave

[14] Austin Andrews, “TriangleIcon.” Accessed: Jan. 23, 2025. [Online]. Available: https://creazilla.com/media/icon/3211438/triangle-wave

[15] Austin Andrews, “SquareIcon.” Accessed: Jan. 24, 2025. [Online]. Available: https://creazilla.com/media/icon/3211095/square-wave

[16] Adam Wathan, “TailwindCSS Documents.” Accessed: Jan. 2025. [Online]. Available: https://tailwindcss.com/docs/

[17] “React Documents.” Accessed: Jan. 2025. [Online]. Available: https://react.dev/learn