CHAPTER 1: INTRODUCTION

1.1 Problem Statement

Effects modules for the guitar, commonly referred to as effect pedals, offer control over an instrument's sound, but are typically unable to create melodies and accompaniment autonomously given a set of limitations.

1.2 Purpose Statement

The goal of this project is to create an effect pedal that uses the guitarist's signal to algorithmically generate unique scales and melodies (a la the modular synthesizer) and send them to various devices over MIDI.

1.3 Context

This project is highly influenced by the design principles of the modular synthesizer. The modular synthesizer, or more simply the modular, is a synthesizer system entirely based on individual *modules*, which each serve a different role in the system. Some modules create sounds from scratch, whether they generate a tone or play a recorded sample. Others modify sounds, such as filters which remove certain frequency bands from sounds. Finally, control modules exist that create voltages by which other modules can be controlled (LeoMakes, 2019). Modules can be selected and interacted with by the user, and are connected with patch cables that route a given signal from one module to another. In this sense, the modular synthesizer is similar to effects pedals in that one can pick and choose what to include in their system.

The synthesizer came to be in the 1960s, with electronic engineers and synth pioneers Don Buchla and Dr. Robert Moog. Moog's synthesizer designs—the first of which being the

Moog Modular Synthesizer in 1964—were intended for use with an orchestra, and featured a full bed of piano-like keys (120years, 2014). Moog's approach to synthesizers as traditional musical instruments became known as *east-coast synthesis*, the counterpart to the more avant-garde *west-coast synthesis* championed by Don Buchla.

Buchla intended to create synthesizers explicitly for the creation of experimental music. His machines did not feature a traditional key bed, and relied on complex *wave-shaping*, a form of *additive synthesis*. Buchla's machines were particularly complex compared to synthesizers being developed on the east coast, and as such a large amount of the nomenclature and development standards for them have by the more simplistic language of east-coast synthesis. However, they have influenced a great deal of musicians and engineers, most notably Moog himself, who said of Buchla, "For the past four decades, Buchla instruments have consistently set the standards for innovative musicians… the panel layouts are nothing short of elegant, while the underlying functions are the most advanced and musically rich available today" (Buchla USA, 2019).

This project is heavily predicated on the idea of *generative modular synthesis*. Generative modular synthesis, or more simply generative modular, is an ideology of musical composition that relies on modular synthesizers to create music. This concept has roots in a genre of early electronic music known as *computer music*—a genre that developed from the use of early computers to algorithmically create music (Britannica, 2015). Generative modular is typically accomplished by setting a number of control modules in tandem such that the sound generation portions of the system will "play themselves"—an illusion created by intelligent systems that can generate sequences, modulate the parameters of sound generation, and perform typical computer

operations such as logic functions and random number generation. The player can then use these controls to set constraints within the system will generate musical phrases and ideas that (ideally) morph over time (Mylarmelodies, 2016).

1.4 Significance of Project

This project will draw inspiration from computer-generated music to aid the guitarist with composition, accompaniment, and sound creation. While modular synths are not inherently incompatible with guitar, they are incredibly costly. The creation of an open-source project that applies certain facets of the composition styles afforded by modular synthesizers to the guitar will be valuable for personal musical purposes while also potentially providing inspiration for professional guitarists and pedal designers.

CHAPTER 2: LITERATURE

2.1 Literature Overview

The research necessary for this project is partially in the realm of electronics, but also in mathematics and physics. A fundamental understanding of how hardware works is essential for the creation of this project, and furthermore an understanding of the mathematics behind music (specifically pitch) is necessary to develop an algorithm that creates music from scratch. Additionally, the project's inspirations are crucial to its context both as a device and as a passion project.

This project likely would not exist if not for the contributions of synth pioneer Bob Moog to the music world. *Moog: A History in Recordings* by Thom Holmes (2016) provides a timeline of Moog and his company's ever-lasting impact on music. Having an understanding and appreciation for the history of synth hardware is crucial to the introduction of new hardware to the market. Moog's impact on music hardware is undeniable and must be credited in such a situation—the culture derived from machines made by Moog and his contemporaries (e.g., Roland, Sequential Circuits) has provided great inspiration for this project. If it weren't for a passionate interest in synthesizers, this project would not exist.

Music played by humans has long been argued to be "robotic" or simply inferior to music generated by computers. Roberto Bresin and Anders Friberg's *Emotional Coloring of Computer-Controlled Music Performances* (2000) attempts to quantify the difference between these two methods of music creation. Bresin and Friberg outline a number of different emotions and how they are quantified by an outside observer, and compare the emotional responses towards each method of music creation. This project aims to programmatically create music that is both

beautiful and human. Using the parameters set forth by Bresin and Friberg (2000), one can quantify how "human" the music created by an algorithm is (and if that level of humanity is consistent).

As this project involves the creation of a melody generation algorithm, having an idea of how other melody-generation algorithms work is greatly beneficial. Z.W. Geem's *Music-Inspired Harmony Search Algorithm: Theory and Applications* (2009) provides an in-depth look at the algorithmic creation of harmonies. Geem's (2009) algorithms are referred to as "meta-heuristic"–defined as using higher-level techniques to find the most optimal value. Geem (2009) includes examples of mathematical algorithms that were tested as well as corresponding pseudo-code for each example. The algorithms are not intended for the creation of the "perfect" harmony, but rather the most optimal harmonies with room for randomization. The purpose of Geem's (2009) research into harmony algorithms is not for the purpose of programmatically creating music; the applications are more geared towards using harmony as the inspiration for optimization problems. Though the applications of Geem's (2009) research are at odds with its relevance to this project, the algorithms developed will provide a great starting point for creating a unique melody generation algorithm.

A key element of understanding how to create scales algorithmically is understanding the mathematical relationships that musical tones have to one another. A series of webpages entitled *Harmony* from Georgia State University's HyperPhysics (n.d.) department outline the mathematical relationships that different notes in an interval have from one another (i.e., a perfect fifth has a 3:2 ratio between its fundamentals). The informational content of *Harmony (n.d.)* is fairly basic, but provides deep insight into the generation of scales. This project will be

determining scales from raw pitch information whose melodic content will be read out in Hertz (Hz). Knowing the math that goes into scale building will allow for less overhead on conversions from pitch to MIDI note–the conversion will only have to be performed once, after the pitches in the scale have been generated.

As such, converting from MIDI note number to fundamental frequency is highly important, and a page from Michigan Technical University's Physics department entitled *Formula For Frequency Table* (n.d.) provides a formula for calculation of pitch for each note in an equal tempered scale based on MIDI note numbers. MTU (n.d.) provides a formula for the determination of the pitch of each note on a standard 88-key piano, though the formula could be used for higher or lower notes above the range of human hearing. While this project's algorithm will not rely entirely on the relationship between note names and fundamental frequencies, being able to convert between the two will be valuable for testing the algorithm. Given a scale output in frequencies, a programmer might not be able to readily determine the viability of the scale. An output of notes, however, can easily be played back on an instrument to determine whether or not the scale is usable. Furthermore, a formula such as MTU's (n.d.) could be used to make sure that notes created by the algorithm are valid notes in the equal tempered scale.

Creating hardware, which will be a significant portion of this project's development cycle, can be daunting for novices. *Handmade Electronic Music: The Art of Hardware Hacking* by Nicolas Collins (2014) provides excellent info on creating music hardware from the perspective of a musician rather than an engineer. Collins (2014) provides advice and example projects to get musicians started creating musical devices with a flair for DIY. Though Collins'

(2014) projects can be fairly basic, they are an excellent starting point for someone with a music background looking to begin creating their own instruments.

Similar to *Handmade Electronic Music* (2014) is a blog post titled *Homemade Guitar Looper with Arduino* by Blogspot user elboulangero (2009) on the creation of a guitar looper pedal with an Arduino. The looper is relatively low-spec, as Arduino is a relatively limited platform in terms of memory. However, eboulangero's (2009) findings contain schematics and a codebase, which will be useful as a resource for interacting with and processing audio on Arduino, the hardware platform on which this project will run.

An important part of any hardware project is choosing a platform. Welch, Wright, Morrow, and Etter's *Teaching Hardware-Based DSP: Theory to Practice* (2002) is meant for instructors to aid in the selection of hardware and software for the teaching of digital signal processing. Though the text is from 2002, much of the software and concepts mentioned in the article are still relevant to this day (e.g., MATLab, C). Despite the fact that the text is meant for instructors, it provides inspiration for this project in terms of examples of relevant platforms and projects to try as a warm-up before delving into the hardware development process.

A unique opportunity afforded by hardware is its ability to influence software–in this project's case, the melody creation algorithm. In a blog post by Emmett Corman titled *Simple Synthesis: Part 11, Sample and Hold* (2017), Corman describes how a sample and hold wave is used in modular synthesis. A sample and hold wave is a waveform that provides pseudo-random voltages over time, which can be used in a number of ways in this project. Aanalog random voltage generators are less stringent than digital random number generators, which could add to the "humanity" of the generated melodies.

The software chosen to prototype and/or run hardware projects can also have a great impact on the project's success. *Pure Data: Another Integrated Computer Music Environment* by Miller Puckette is a 1996 document on then-upcoming visual programming language Pure Data. Pure Data (or Pd) is a language designed in competition with Max, a common platform for digital synthesizers and effects. Though Puckette's (1996) paper is twenty-three years old, Pd's design philosophy has remained consistent throughout its life cycle; in addition to this, *Pure Data* (1996) is an overview of the language's potential rather than a piece of documentation, so its age is less problematic. Pd's objects are written in C (or optionally C++), meaning that any algorithms or code written for this project can be easily ported to Pd for prototyping.

2.2 Hardware

Two related pieces of hardware are also major inspirations for the project. Chase Bliss Audio's *Thermae* analog delay pedal provides sequenced digital control over the delay time of the effect, creating harmonically relevant melodies to the input signal. The melodies that *Thermae* generates are the same as whatever notes were played into the pedal, but sped up so as to appear as melodies a fourth, fith, octave, and more above or below the input signal ("Thermae", 2018).

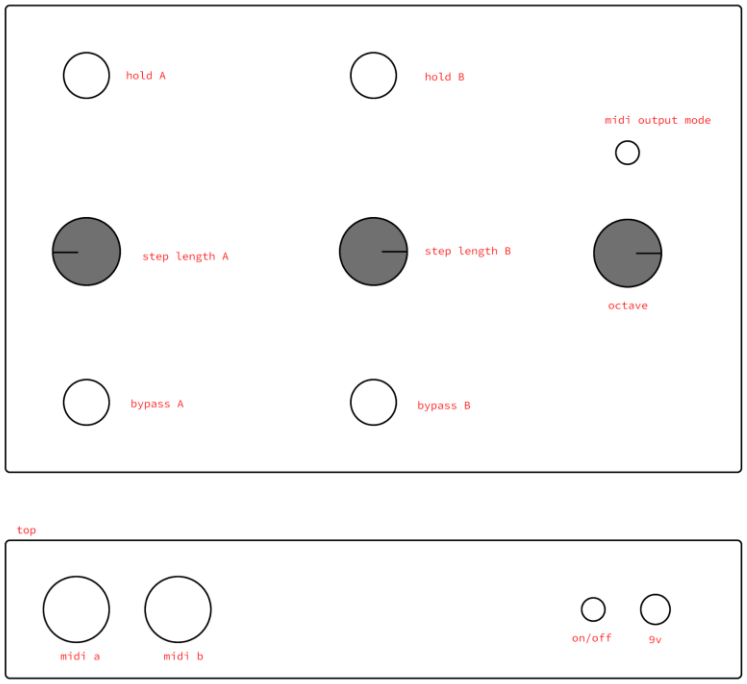
Another related piece of hardware is the Music Thing Modular *Turing Machine*. The Turing Machine creates no sound of its own. Instead, it creates stepped random voltages based on a tempo. Voltages from the Turing Machine can be sent to other synth modules via a patch cable; the voltages are then used to create melodies and modulate parameters throughout the system. When the user is satisfied with a random sequence, they can lock the sequence in place

so that it plays back on repeat. The Turing Machine's random generation is completely random, with no basis on an input pitch (Music Thing Modular, n.d.).

CHAPTER 3: METHODS

Design

The design spec of this project is such that the final product will be a device similar to a modular synthesizer's control modules in a guitar pedal format. It will be a physical device that fits into a standard guitar pedal enclosure—specifically, the Hammond 1590DD. The layout of the interface (as of December 2019) is as follows:



The design philosophy of this interface is that everything is large and spaced out properly. Footswitches, such as those used for bypass and hold, need to be spaced far apart from one another such that someone in a performance situation can activate each switch without worry of accidentally pressing the wrong switch. Knobs controlling constraints for melody playback are

all in the middle of the device or near another relevant control. MIDI jacks are aligned with respect to each control lane.

Frameworks

This project will run on Arduino (more specifically Teensy, a platform based on Arduino), and C++. Arduino is the most popular microcontroller on the market, making it a good choice for a novice in hardware development. There are a great deal of resources on the internet for Arduino such as blogs, forums like StackOverflow, and great documentation. Running C++ on the Arduino is necessary over C, as C++ has a bigger standard library, meaning that a lot of the distractions (e.g., the creation of data structures like linked lists) of building an algorithm for melody generation are taken care of immediately.

Algorithms

The majority of the project's codebase will contain or otherwise support the melody generation algorithm. The algorithm takes two input parameters, an array of the notes that the scale is to be based on, and the desired size of the output scale. The algorithm then determines the number of remaining notes available in the octave. For example, if the notes A5, C#5, and D5 are given to the algorithm, it will determine that A#5, B5, C5, D#5, and so on until A#6 (exclusive) are available for use in the scale. The algorithm will then test each available note, finding the average *harmony coefficient*—a comparison of two notes based on the ratio between their fundamental frequencies—of the given note and the current scale. The algorithm will then determine which note has the best average harmony coefficient, add it to the scale, and then start

the process again using those notes. It will continue this process until the scale reaches the desired size. Before being returned, the scale will be checked as to whether or not it is valid (i.e., a standard heptatonic scale ought to have one of each named note, be it sharp or flat). The algorithm in its current state (as of December 2019) can be represented in pseudocode as such:

```
Note[] generateScale(Note[] basis, int size){

     // we can assume that xorScale finds which notes
     // are not in any given scale

     Note[] missing = xorScale(basis)
     Note[] scale = basis

     while (scale.size != size){

          // gather all scores from missing notes
          dict<Note, int> scores = {}

          for note in missing{

               // determine how well note fits in scale
               // and store the result
               dict[note] = getAvgHarmony(note, scale)

          }

          // determineBestScore() gets best harmony
          // score and returns corresponding Note
          scale.append(determineBestScore(scores))

          // validate scale--check if all note names
          // are accounted for and removes offenders
          if(scale.size == size){
               validateScale(scale)
          }
     }
     return scale
}
```

Analytical Methods

As this project is designed to create music, analyzing its output will be difficult. From an objective standpoint, the project can be judged in the following manner:

1. Does the device create randomly generated melodies?
2. Are those melodies created from notes sampled from the user's playing?
3. If given constraints (e.g., octave range, step length, repeated sequences), does the device follow those constraints?

If the device checks all of these boxes, then it can be considered to be a working product. However, a product that works is not necessarily a product worth using. Given that this project is designed to create music, analysis of the music it creates is a more personal, philosophical question than it is a Boolean *yes* or *no*.

To call into question the intrinsic worth of any musical work is to invoke an ontological and epistemological discussion of what music is and how we understand it. A common fundamentalist ontology of music as outlined in the Stanford Encyclopedia of Philosophy's *The Philosophy of Music* (2017) is a *performance theory*—that works of music can be understood as actions. In Aitor Izagirre and Igor Petralanda's *On the Idea of Action in Davies' Performance Theory of Art* (n.d.), Izagirre and Petralanda explain this to mean that a work of music is entirely abstract until it is performed, whereupon the choices made by the performer (intentional or not) define it as a concrete work.

The Philosophy of Music (2017) also grants that in most theories of determination of the value of a given musical work, emotional response plays a large role. Tying into performance theory, this means that the emotional response created by certain actions taken by a musician are

the basis for its worth. In her landmark work, *Elements of Expression in Music*, Kate Hevner (1936) developed what is known as the *adjective circle*, a non-comprehensive list of adjectives that can be used to describe the emotional response that a given listener might have upon hearing a musical work. This adjective circle can, for the purposes of this project, serve as a way to quantify emotional response to music.

Using performance theory as our basis for what musical works are and emotional response In order to judge the worth of the music created by this project, what an *action* is in regards to music created by the device itself must be defined. Within the parameters set for a given performance, the device can make choices on the following:

- The scale to be played in
- What note to play when
- The velocity of the given note

As these three points are the only influence the device itself has over the music being played, these can be determined to be the actions it can take during any given performance. These actions must be judged in the context of a given performance. In an improvisational performance, the music created by the device can be judged on whether or not it evokes any kind of emotional response. In a performance of a song, the music created by the device can be judged on the emotional response created by the improvised playing in tandem with a structured performance.

Features

The core features of the project are as follows:

- The device will receive an input signal and output it unchanged.
- The device will read and analyze the input signal for the root frequency of the note/chord being played.
- The device will build an equal-tempered melodic scale based on the notes gathered from the input signal.
- The device will send random notes within that scale via MIDI based on a tempo set by the user.
- The user may trigger the device such that one MIDI channel plays the last *x* notes again, where *x* is the step length (of a range from 1-16) set by the user.

These features are essential for the product to function as a whole. The following are stretch goals to be implemented should there be time:

- The user may select a MIDI output mode such that MIDI channels A and B play:
    - the same notes in parallel (and can be bypassed independently).
    - different notes in parallel (and can be bypassed independently).
    - the same notes, but only one channel selected by the user plays at a time.
    - the same notes and alternate MIDI output from A to B per step such that output of each channel is mutually exclusive—if one output is playing a note, the other is silent.
- The user may trigger the device such that *both* MIDI channels play the last x_A or x_B notes again, where x_A and x_B are the step length (of a range from 1-16) set by the user for channels A and B, respectively.
- The device will play a third melody on a built-in synthesizer circuit.

- The device will use a sample and hold circuit to influence externally what notes to play and when.
- The device will output its pitch data in 1 volt per octave-tuned modular synthesizer control voltage.

Test Plan

The melody generation algorithm will be tested both mathematically and qualitatively. The music produced by this device will be judged using the performance theory-based paradigm for judging the worth of a piece of music discussed in the *Analytical Methods* section of this chapter, which will be analyzed in correspondence with mathematical tests. To test the mathematical validity of a scale, the device will determine and output to a console the average harmony coefficient of the scale. Lower average harmony coefficients indicate a more harmonious scale. Scales will then be evaluated by ear to determine if they are musically viable. A developer will take notes using Hevner's (1939) adjective scale on what kind of emotions a given scale evokes, afterwards marking each adjective scale with its harmony coefficient.

Criteria and Constraints

One major consideration when creating a device that algorithmically generates music is to make sure that the rules of the algorithm are not too hard and fast. This is a large part of the decision for the device to play melodies randomly as opposed to another alternative such as generating them with machine learning—if a melody-making algorithm is too targeted, too specific, it runs the risk of creating existing, copyrighted melodies. If this were to happen, the

project would run the risk of being taken down from websites like GitHub, YouTube, or other social media via a copyright claim.

References

120years. (2014, April 3). 'Moog Synthesisers' Robert Moog. USA, 1964. Retrieved from https://120years.net/moog-synthesisersrobert-moogusa1963-2/

Bresin, R., & Friberg, A. (2000). Emotional Coloring of Computer-Controlled Music Performances. *Computer Music Journal*, *24*(4), 44–63. doi: 10.1162/014892600559515

Britannica. (2015, May 13). Computer music. Retrieved from https://www.britannica.com/art/computer-music

Buchla USA. (2019). History. Retrieved from https://buchla.com/history/

Collins, N. (2014). *Handmade Electronic Music: the Art of Hardware Hacking*. Hoboken: Taylor and Francis.

Corman, E. (2017). Simple Synthesis: Part 11, Sample and Hold. Retrieved October 11, 2019, from https://www.keithmcmillen.com/blog/simple-synthesis-part-11-sample-and-hold/.

elboulangero. (2009). Homemade guitar looper with Arduino. Retrieved October 11, 2019, from http://arduino-guitarlooper.blogspot.com/2009/10/introduction-here-how-to-produce-small.html.

Geem, Z. W. (2009). *Music-inspired harmony search algorithm: theory and applications*. Berlin: Springer.

Georgia State University. (n.d.). Musical Scales. Retrieved October 11, 2019, from http://hyperphysics.phy-astr.gsu.edu/hbase/Music/mussca.html.

Hevner, K. (1936). Experimental Studies of the Elements of Expression in Music. *The American Journal of Psychology, 48*(2), 246-268. doi:10.2307/1415746

Holmes, T. (2016). Moog: A History in Recordings . Retrieved October 11, 2019, from https://moogfoundation.org/moog-a-history-in-recordings-by-thom-holmes/.

Izagirre, A., Petralanda I. (n.d.). *On the idea of Action in Davies' Performance Theory of Art*. (Doctoral dissertation, The University of the Basque Country, Greater Bilbao, Basque Country, Spain). Retrieved from http://www.gabone.info/txt/izagirre_the-idea-of-action-in-performance-theory-of-art.pdf

Kania, A. (2017, July 11). The Philosophy of Music. Retrieved December 8, 2019, from https://plato.stanford.edu/entries/music/.

LeoMakes. (2019, April 26). What exactly is a Modular Synth? Retrieved from https://www.attackmagazine.com/technique/technique-modular-synthesis/what-exactly-is-a-modular-synth/

Michigan Technical University. (n.d.). Formula for frequency table. Retrieved October 11, 2019, from https://pages.mtu.edu/~suits/NoteFreqCalcs.html.

Music Thing Modular. (n.d.). 22 things to know about the Turing Machine. Retrieved October 11, 2019, from https://musicthing.co.uk/pages/turing.html.

Mylarmelodies. (2016, October 31). *An Intro to Making Generative Music on Modular.* Retrieved from https://www.youtube.com/watch?v=NvrxQbh6vAg.

Puckette, M. S. (1996). Pure Data: Another Integrated Computer Music Environment. *Second Intercollege Computer Music Concerts*, 37–41.

"Thermae: Analog Delay / Pitch Shifter || Mini-Doc." *YouTube*, uploaded by Chase Bliss Audio, 10 May 2018. https://www.youtube.com/watch?v=Zg70GAALnOY

Wright, C. H. G., Welch, T. B., Etter, D. M., & Morrow, M. G. (2002). Teaching hardware-based DSP: Theory to practice. *IEEE International Conference on Acoustics Speech and Signal Processing*. doi: 10.1109/icassp.2002.5745571