Musical Tuning Systems as a Form of Expression

Project Team:

Lewis Cook ljcook@wpi.edu

Benjamin M’Sadoques bgmsadoques@wpi.edu

Project Advisor

Professor Wilson Wong

Department of Computer Science

Abstract

Many cultures and time periods throughout history have used a myriad of different practices to tune their instruments, perform, and create music. However, most musicians in the western world will only experience 12-tone equal temperament a represented by the keys on a piano. We want musicians to recognize that choosing a tuning system is a form of musical expression. The goal of this project was to help musicians of any skill-level experience, perform, and create music involving tuning systems. We created software to allow musicians to experiment and implement alternative tuning systems into their own music.

Table of Contents

Table of Figures

1 Introduction

1.1 The Problem

12-note equal temperament is likely the only set of tones most western musicians will ever experience, play, and write with. Writing music is difficult, even using a familiar musical notation. Currently most musicians are unfamiliar with anything beyond modern 12-note equal temperament [1]. While this is not a problem, there is a whole world of alternative tuning systems (referred to as “ATS” in this paper), that many musicians will never experience. These tuning systems are also sometimes referred to as microtonal music, xenharmonic music, temperaments, among many other names.

12-Note Equal Temperament is a tuning system that has 12, evenly spaced notes for every octave. A tuning system is simply a collection of tones. A tone represents a complex waveform with a base frequency in Hz [2]. In tuning systems, a note is simply a tone that is given a name. The tuning system defines the frequency value for each note, and the names for each note. Furthermore, most tuning systems contain octaves which are the space between two notes where the second note has double the frequency value of the first [3]. Both notes that define an octave are considered the same note. For example, in 12-Note Equal Temperament, the frequencies 220hz and 440hz are both defined as “A” notes and the space between the two is considered an octave. All the other notes in an ATS will exist inside an octave, the octave will dictate whether a note will sound higher or lower than its counterpart notes in other octaves. While this may seem simple to experienced musicians, understanding the foundations of the terminology music theoreticians use is critical to understanding and using ATS.

Music is all about exploration, experimentation; to make something no one has ever heard before. While most musicians are perfectly fine with using 12-Note Equal Temperament, we believe that other systems should not go unnoticed. Equal temperament is just one tuning system out of the infinite possibilities available to musicians. In fact, Equal Temperament did not exist until it was invented by Zhu Zaiyu in 1596 [1]. Instead, most musicians worked with other ATS such as Meantone or Pythagorean Temperament [1].

Some musicians view music theory as an arduous, complex set of rules that strip away creativity. Despite this notion, every musician uses some form of theory whether they know it or not. You do not need to know every grammar rule that exists to hold a conversation or write a paper. Similarly, you do not need to be a music theory expert to create good music. Music theory is based in mathematics; it may seem very specific, but it is also flexible. This mechanic is proven by the myriad of musicians who can seamlessly break the “rules” of music theory, and still write incredible pieces, such as Ornette Coleman with his signature free jazz sound [4]. On the other hand, there are musicians who know little to no music theory but can create amazing music simply through their ears such as Jimmy Hendrix who famously never formally studied music theory [5]. This proves that music theory is not a strict set of rules that one must conform to, but it can give musicians a sense of direction rather than playing random notes.

Just as a high-level understanding of music theory is not critical to create music, experimenting with ATS should not be restricted to music theory experts. Many instruments, especially stringed ones, can produce almost every tone imaginable but without proper guidance, such as frets, muscle memory, or in the case of a brass instrument, valves, it is difficult to know how to play a specific tone. Playing tones outside 12-Note Equal Temperament is easy to do, but difficult to control. We take 12-Note Equal Temperament for granted; musicians almost never have to wonder if a tone fits the temperament system they are playing. Everything outside 12-Note Equal Temperament is out of tune, rather than a new tone to use. Without a simple avenue or tool to experiment or try out new ATS, most musicians will never explore beyond what they know. We seek to introduce ATS to musicians around the world.

1.2 Goals

The goal of this project was to create a tool for musicians to explore and experiment with ATS that they likely could not control before. To accomplish goal, we created a VST (virtual studio tool) plugin for DAWs (digital audio workstations) that gives musicians an easy and simple way to use ATS in their music. First, the plugin allows users to select from a list of pre-existing ATS, connect with any virtual or physical MIDI (Musical Instrument Digital Interface) device, and play the tones that exist in the ATS. The user can seamlessly use the ATS for writing and recording music. Second, the plugin gives the user information on the ATS, so the user can understand why an ATS is the way it is. The plugin displays what the ATS is, its origins, simple music theory tips, and how it is constructed. Third, for the truly adventurous, the user can create and store their own unique ATS for the ultimate form of self-expression and experimentation.

2 Research

2.1 Background

2.1.1 Music Terminology

Music theory is an incredibly complex subject that one could spend an entire life researching. However, music theory basics are simple concepts most musicians, no matter the skill level, are aware of. However, it is important to establish a baseline of terms that are important to understand.

- **Tone**: A tone is a complex waveform that we perceive as a single base frequency or pitch [2]. Frequency in music is the speed of the vibration of a sound wave (figure 1). Higher frequency will result in a higher tone, while lower frequency results in a lower tone. Tuning systems defines specific tones as notes. For example, in equal temperament, a middle C will always be tuned to approximately 261.63 Hz.

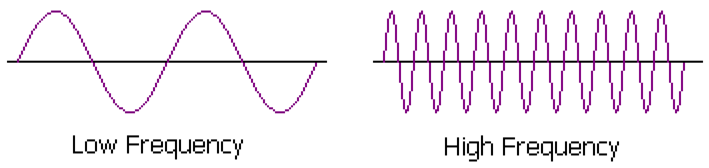


Figure 1 Frequency in Music

- **Octave**: An octave is the distance of frequency between two notes of the same type. The last note in the octave will have double the frequency value as the starting note in the octave and both will share the same name [3]. The octave serves as a neutral sounding interval to base systems on. For example, the two tones that occupy 220 Hz and 440 Hz form an octave. In equal temperament, the 220 Hz and 440 Hz pitches are both called A. The notes in an octave will be repeated throughout the entire system as shown on a piano key layout (figure 2).

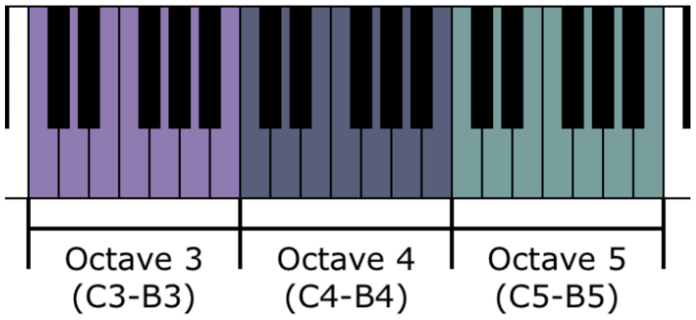


Figure 2 Octave in 12 note equal temperament represented on a keyboard.

- **Scale**: A scale is a specific selection of tones that usually exists inside of one octave. The tones of scales are played melodically, or consecutively [2]. All the tones in a scale are meant to sound good together. For example, in equal temperament, the “C Major” scale consists of the notes, C, D, E, F, G, A, and B (figure 3). The two most common types of scales are the Major and Minor scales, the former sounds “happy” while the latter sounds “sad”.

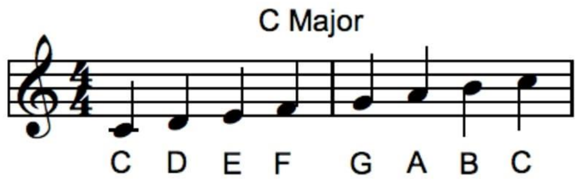


Figure 3 C Major scale represented on a musical staff.

- **Interval**: An interval is the distance in frequency between two tones, usually within the same ATS [2]. For example, a "fifth" interval is the ratio between the first note of a scale and the fifth note of that scale. A fifth interval in the C Major Scale is constructed with the C and G notes. Intervals are often represented as ratios that encode the difference between tones rather than the exact frequencies. For example, the interval 220 to 440 can be represented as the ratio 2/1. Here are some important types of intervals.
    - **Just Intervals**: A just interval is when the difference between tones is based on whole number ratios. Just intervals sound clean and pure compared to tempered intervals. Some common just intervals are 1/2 (octave), 3/2 (perfect fifth), 4/3 (fourth) [2]. In waveforms, the different tones match up on top of one another over a short time (figure 4).

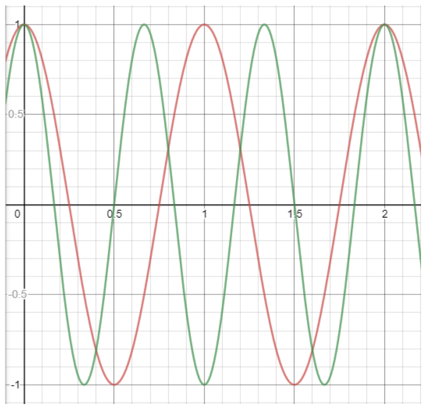


Figure 4 Just Intonation Waveform ratio 3:2

    - **Tempered Intervals**: A tempered interval is an altered whole numbered ratio interval. They are usually intended to remove wolf intervals (intervals that are considered out of tune and sound dissonant) from ATS, or to fit the system to a man-made layout, such as a piano [2]. In waveforms, the different tones do not match up exactly (like just intonation), but they are quite close (figure 5). The ratio complexity will be the same as the just intonation interval, but the interval will sound slightly off to trained ears.

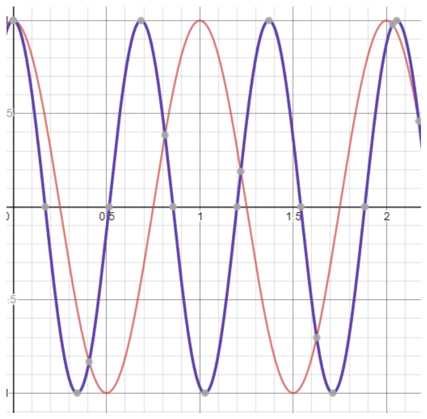


Figure 5 Tempered Interval Waveform 1.46

- **Interval complexity**: Interval complexity is based on what harmonic an interval is and describes the sound of an interval. The complexity is represented by the second number in any ratio. As the ratios get more complex the period (the time it takes to complete one waveform cycle) between wave forms gets longer. For example, 4/3 has a short period (figure 6).

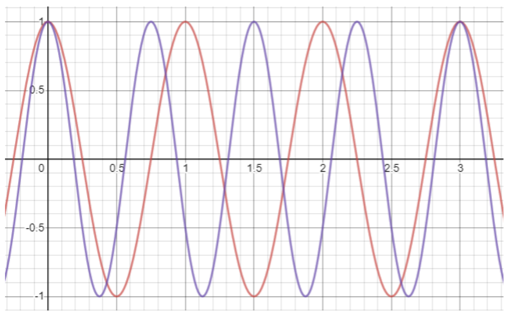


Figure 6 Mild interval complexity 4/3

17/12 has a far longer period compared to 4/3 (figure 7).

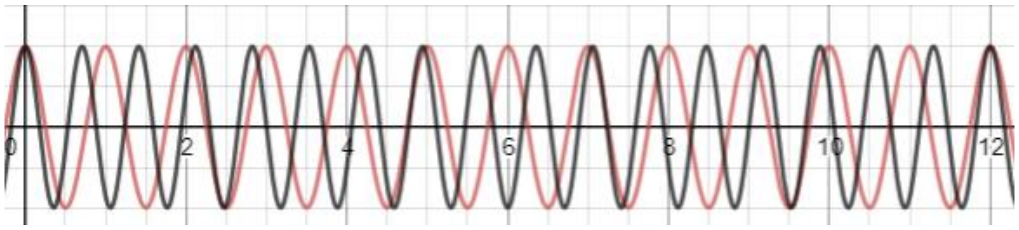


Figure 7 Complex interval 17/12

Even if the numerator changes, the period stays the same. 13/12 is a different ratio but the complexity is the same (figure 8).

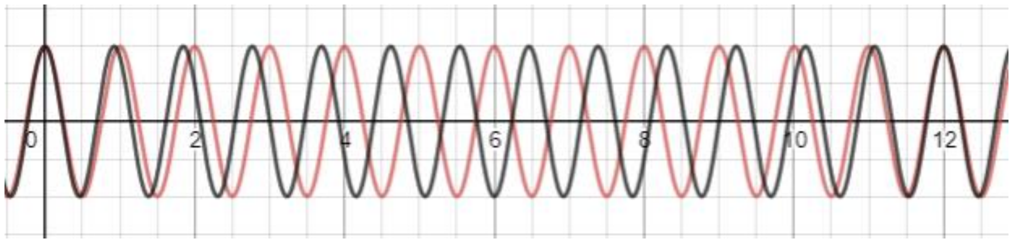


Figure 8 Complex Interval 13/12

- **Wolf Intervals**: Wolf Intervals are considered too far out of tune for musical performance due to their complex ratios (figure 9). They are untuned intervals left-over from tuning systems that do not tune these intervals directly [2]. Wolf intervals will always sound impure to a trained ear. The existence of wolf intervals is directly responsible for historical experimentations with ATS.

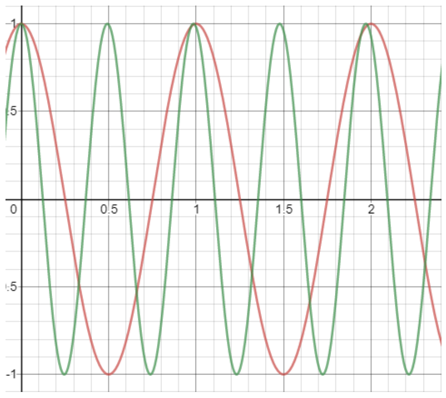


Figure 9 Wolf Waveform, 2.03 an altered octave

- **Cents**: A cent is a logarithmic unit of measurement for music intervals [6]. In 12 note equal temperament, each semitone or interval is 100 cents apart from each other.
- **Chords**: A chord is a set of tones meant to be played harmonically, or simultaneously [2]. For example, in equal temperament, a C major chord consists of the notes C, E, and G being played harmonically. Chords can use any number of notes.
- **Key Color Contrasts**: A circumstance that happens in any unequal temperament. Each scale contains its own unique qualities or effects. The more unequal a temperament the greater the contrasts. For example, in meantone temperament we can characterize the change by looking at the major thirds, which vary in size depending on the key played [2].
- **Timbre**: Timbre is purely how the instrument sounds, regardless of the tone played [7]. For example, a guitar will have a different timbre than a piano even when they both play a middle C note.

Music theory features many more music terms, but everyone should have a baseline understanding of these terms to understand our project. When it comes to modern music production, there are many contemporary musical tools that must be understood. The most important is the DAW. A DAW (digital audio workstation) is a piece of software that musicians use to produce recorded music. From initial development to finishing touches, almost everything happens inside of a DAW. Some of the most common DAWs include Garage Band, Ableton, FL Studio, Pro Tools, and Reaper [8]. A VST (virtual studio tool) is a type of plugin that works inside most DAWs. VSTs can range from effects one adds to their instruments, such as a reverb effect, to instruments themselves, such as a drum machine or synthesizer. Most VSTs have their own user interface and controls, they can input and output MIDI files and audio data. MIDI (musical instrument digital interface) refers to the communication system between a controller and a computer used to produce music in real time. MIDI is the most common way to create or synthesize music notes on a computer. MIDI is either constructed manually in the DAW or performed on a MIDI controller. Many controllers are formatted to look like keyboards. For example, one could create a MIDI track to the melody of "Happy Birthday" controlling the length, volume, and pitch of each note. Then, using that track, one could apply any digital instrument and have that instrument play "Happy Birthday".

2.1.3 Tuning Systems

The project team decided to implement six different ATS into the plugin. The team chose a culturally and historically diverse set of ATS so the user can try out a relatively wide variety of ATS.

2.1.2.1 Equal Temperament

Equal temperament represents the full compromise of making tuning systems functional and consistent rather than using pure Just intervals. As a result, there are no wolf intervals, and every half-step is consistent. Unlike other ATS, Equal Temperament has no variations. Equal Temperament divides the octave into any number, usually 12, of equally spaced half steps, each half step is based on the tempered interval $2^{1/12}$. [9]. The half step interval can be changed into a simple function that can be used to create any equal temperament. $f(b, n) = b^{1/n}$, where b is the base, previously 2, and n is the octave size, previously 12. B and n can be any positive rational number.

2.1.2.2 Just Intonation

Just intonation itself is not technically an ATS. There is no single Just Intonation tuning system, instead it represents a harmonic idea to build scales based on whole number ratios or just intervals rather than tempered intervals. This is unlike many other ATS such as Equal Temperament, Well Temperament, or Meantone Temperament. This method creates a functional system of ratios that are easy to tune but limit the number of intervals that can be performed. Musicians can build a Just Intonation ATS to have a specific major scale, or natural resolutions, to be functional rather than flexible [10]. Acoustic ratios work based on complexity, that the higher the denominator is, the more dissonant the ratio will sound.

Tone name	ratios	Frequency (in Hz)
A	1/1	440
A-sharp/B-flat	16/15	469.33333
B	9/8	495.00000
C	6/5	528.00000
C-sharp/D-flat	5/4	550.00000
D	4/3	586.66667
D-sharp/E-flat	17/12	623.33333
E	3/2	660.00000
F	8/5	704.00000
F-sharp/G-flat	5/3	733.33333
G	16/9	782.22222
G-sharp/A-flat	15/8	825.00000
A	2/1	880

Figure 10 12-tone Base 2 Just Intonation example

2.1.2.3 Meantone Temperament

At around the 1700's, musicians in Europe were conflicted. One philosophy was to base the tuning on pure, pretty-sounding Just Intonation intervals, and another, was to temper systems to make them more usable. Meantone Temperament was one of the first types of ATS to fit this compromise. There is no one Meantone Temperament ATS, there were many solutions to reducing wolf intervals. Wolf intervals were left-over intervals from tuning Just Intonations, they were considered too out-of-tune for performance and can sound harsh and unnerving. Meantone Temperament reduced wolf intervals by tuning specific tones to be mean or average between other tones. For example, C is mean between B-flat and D, where B-flat is below C [9].

Tones	half-step ratios	Tones (in Hz)
A	1	440
A-sharp/B-flat	1.068168972	469.9943479
B	1.048233588	492.6638614
C	1.068232322	526.2794604
C-sharp/D-flat	1.048185725	551.6386178
D	1.066901626	588.5441383
D-sharp/E-flat	1.067951509	628.5366005
E	1.047817897	658.5918986
F	1.068314717	703.5834181
F-sharp/G-flat	1.048057146	737.3956294
G	1.066731285	786.6029874
G-sharp/A-flat	1.04835824	824.6417232
A	1.067130094	880

Figure 11 17th-centry Irregular Equal-beating mean-tone Temperament

2.1.2.4 Well Temperament

Well Temperament was developed after Meantone Temperament to allow musicians to play in all keys. This ATS was another compromise from the pure tones of Just Intonation, and more tempered tones of Meantone Temperament, but not yet a full compromise like Equal Temperament. Well Temperament eliminated wolf intervals, and allowed musicians to perform in all the keys, with key-color contrasts. These key-color contrasts, or "character of the keys", happened because each scale contains its own unique qualities or intervals. Different tuners could create variations on Well Temperament by tempering the spacing of the thirds. Some musicians confuse Well Temperament with Equal temperament [9].

Tones	half-step ratios	Tones (in Hz)
A	1	440
A-sharp/B-flat	1.063329484	467.8649728
B	1.053867457	493.0676691
C	1.065212939	525.222061
C-sharp/D-flat	1.056126862	554.7011272
D	1.060280738	588.1389207
D-sharp/E-flat	1.060667713	623.819964
E	1.056227518	658.8958121
F	1.065111427	701.7974588
F-sharp/G-flat	1.053867457	739.6015033
G	1.062271339	785.6574792
G-sharp/A-flat	1.059051447	832.0516903
A	1.0576266	880

Figure 12 Francesco Antonio Vallotti's Early Eighteenth-Century Well Temperament

2.1.2.5 Maqam

Maqamat (plural of Maqam) are scale-like abstractions from Turkish, Arabic, and Persian music. Historically, Arabic and Turkish music have over 50 different Maqamat each, while Persian music has over 10 [11]. All the Maqamat versions are slight variations on one another. Most Modern-day Maqamat simply use a 24-note equal temperament scale which is like 12-Note Equal Temperament, just with added microtones between the notes

[11]. Despite this “standardized” version, many composers will often emphasize that certain notes should be a tad higher or lower than the note’s standard position [12].

2.1.2.6 Gamelan

Gamelan is the traditional music ensemble of the Sudanese, Balinese, and Javanese people of Indonesia. Gamelan is mainly played through gongs, or other tuned metal instruments struck with mallets. Gamelan instruments are generally tuned to one of two ATS, Slendro and Pelog. These ATS vary based on location and ensemble, but they still loosely follow a replicable formula [13].

For all intents and purposes, the Slendro system is a 5-note equal temperament (5ET) [14]. Meaning it uses equal temperament, but it splits the octave into 5 notes instead of 12. Traditionally, the intervals would not be exact, but they are roughly even to one another. The closest western approximation would be the pentatonic scale which consists of A, C, D, G, and E. However, in Slendro, the C, D, and G notes are tuned noticeably lower than the western pentatonic scale. The notes of the Slendro scales are commonly referred to as simply 1, 2, 3, 4, and 5.

Pelog is different from Slendro because it has 7 notes instead of 5. The seven notes have unequal intervals between them. Furthermore, all 7 notes are not generally used. There are three “modes” of pelog which select 5 notes out of the 7 to use, the other two notes being used as embellishment [15]. These variations originate from different locations in Indonesia. The 5 notes selected are referred to as (in order) dong, deng, dung, dang, and ding. There is not a “standard” when it comes to tuning pelog as many of the different sub-systems tune slightly different. Some music theorists claim that Pelog tuning is more about the “feel” of the notes rather than a rigorous mathematical system. However, there is note pitch documentation and research on specific gamelan ensembles which is the information we can use for the program [16].

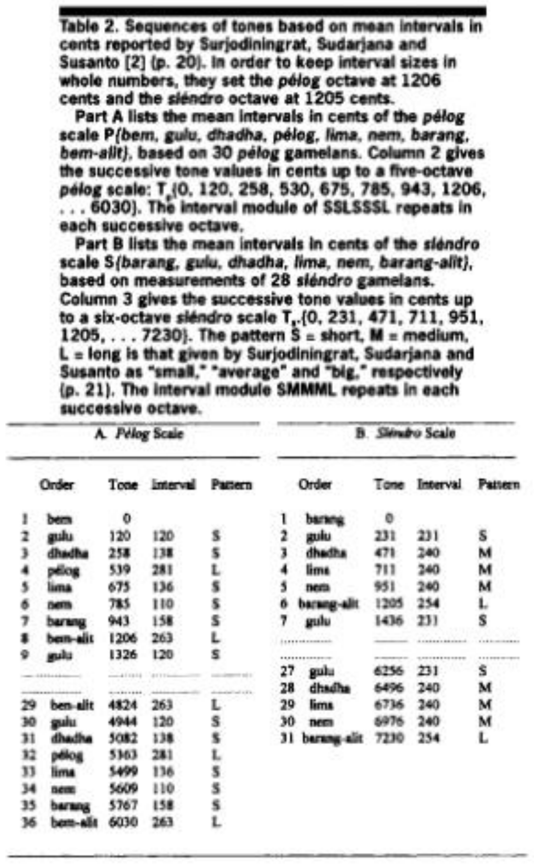


Table 2. Sequences of tones based on mean intervals in cents reported by Surjodiningrat, Sudarjana and Susanto [2] (p. 20). In order to keep interval sizes in whole numbers, they set the *pélog* octave at 1206 cents and the *sléndro* octave at 1205 cents.

Part A lists the mean intervals in cents of the *pélog* scale P{*bem, gulu, dhadha, pélog, lima, nem, barang, bem-alit*}, based on 30 *pélog* gamelans. Column 2 gives the successive tone values in cents up to a five-octave *pélog* scale: T_p{0, 120, 258, 530, 675, 785, 943, 1206, . . . 6030}. The interval module of SSLSSSL repeats in each successive octave.

Part B lists the mean intervals in cents of the *sléndro* scale S{*barang, gulu, dhadha, lima, nem, barang-alit*}, based on measurements of 28 *sléndro* gamelans. Column 3 gives the successive tone values in cents up to a six-octave *sléndro* scale T_s{0, 231, 471, 711, 951, 1205, . . . 7230}. The pattern S = short, M = medium, L = long is that given by Surjodiningrat, Sudarjana and Susanto as "small," "average" and "big," respectively (p. 21). The interval module SMMML repeats in each successive octave.

A. *Pélog* Scale

Order		Tone	Interval	Pattern
1	bem	0		
2	gulu	120	120	S
3	dhadha	258	138	S
4	pélog	539	281	L
5	lima	675	136	S
6	nem	785	110	S
7	barang	943	158	S
8	bem-alit	1206	263	L
9	gulu	1326	120	S
………		………	………	………
………		………	………	………
29	ben-alit	4824	263	L
30	gulu	4944	120	S
31	dhadha	5082	138	S
32	pélog	5363	281	L
33	lima	5499	136	S
34	nem	5609	110	S
35	barang	5767	158	S
36	bem-alit	6030	263	L

B. *Sléndro* Scale

Order		Tone	Interval	Pattern
1	barang	0		
2	gulu	231	231	S
3	dhadha	471	240	M
4	lima	711	240	M
5	nem	951	240	M
6	barang-alit	1205	254	L
7	gulu	1436	231	S
………		………	………	………
………		………	………	………
27	gulu	6256	231	S
28	dhadha	6496	240	M
29	lima	6736	240	M
30	nem	6976	240	M
31	barang-alit	7230	254	L

Figure 13 Gamelan Pelog and Slendro Intervals

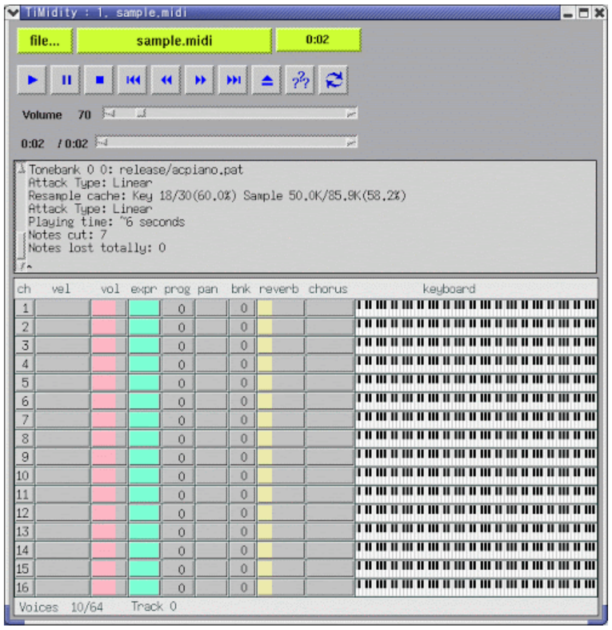
2.1.2.7 Harry Partch Scale

The Harry Partch 43 Tone scale is an experimental ATS developed and created by Harry Partch in the mid-1900s. The Harry Partch scale takes influence from Just Intonation as it uses whole ratios. However, this scale has 43 notes per octave, an incredible number. Music for this ATS has historically been created through multiple different instruments. The plugin has the capacity to play the entire Partch scale as one instrument. Harry Partch started off using an "11-Limit Tonality Diamond" to determine the main notes of his scale, this means that all the interval ratios are rational numbers with where the denominator does not exceed 11. He noticed that there were big gaps between certain notes, so he went beyond the Tonality Diamond to fill in the gaps with additional filler notes. These notes still adhered to whole number ratios.

2.2 Literature Review: Existing Programs

2.2.1 Virtual Studio Tools

ATS are generally a niche topic, even among musicians. Therefore, there are not too many existing VSTs that work with ATS. Some programs like TiMidity++, expect the user to already be knowledgeable on the subject to even use the program because it does not provide much instruction or guidance for first time users. Furthermore, TiMidity++'s user interface is incredibly confusing to someone who is an expert on music theory. Not purely for creating music with ATS, TiMidity++ is a MIDI conversion synthesizer that allows users to tune their own MIDI files to alternative temperaments.


TiMidity : 1. sample.midi
file...
sample.midi
0:02
Volume 70
0:02 / 0:02
ch vel vol expr prog pan bnk reverb chorus keyboard

```
Tonebank 0 0: release/acpiano.pat
Attack Type: Linear
Resample cache: Key 18/30(60.0%) Sample 50.0K/85.9K(58.2%)
Attack Type: Linear
Playing time: ~6 seconds
Notes cut: 7
Notes lost totally: 0
```

```
Voices  10/64     Track  0
```

Figure 14 TiMidity++ User Interface

VSTs such as Chipsound have microtonal capabilities and essentially allow the user to create their own ATS [17]. However, Chipsound is designed to create chiptune music, so it has a limited variety of timbres, but it has microtonal capabilities. This problem is apparent with most VSTs with some sort of microtonal aspects. They add the ATS features as an optional bonus, instead of it being the focus.

2.2.2 Other Program Formats

Our program is a VST plugin, but other program formats were researched for new ideas. Each app was examined to understand how accessible the program was, how programs helped the user understand the program, the support for ATS, and the ability to connect, share, and save musician’s work.

Accessibility is a top priority for this project, to achieve this other programs’ accessibility must be examined and critiqued. Many of the apps examined were accessible, but most had some major drawbacks. A few programs, such as microsound, rationale, and

Scala, required the user to manually download other software. This practice may make the process confusing for users to fully download and install applications and plugins [18-20]. All the programs were downloaded with Norton 360 (Anti-virus) enabled and windows defender. Some programs, such as Csound, Mutabor, Sound Modeler, and Scala, triggered the anti-virus in some way [20-23]. It is unknown how the programs trigger the anti-virus. A few programs, such as Offtonic Microtonal Synthesizer, and Scale Workshop, were web applications that required few downloads. This practice made the easiest apps to access [24, 25]. A few apps, such as, Blue, Hex, Melodyne 5, Schismata, Tune Smithy, provided a simple download process, either through a one-click-installer, or full-program zip file [26-29]. None of these programs triggered the anti-virus. Accessible programs should have a simple install process, provide necessary dependencies, and should not trigger ani-virus warnings.

Other applications will showcase to make plugins understandable to users. Some apps, such as, Mutabor, Rationale, and Tonescape, had little or no documentation on how to use the program [19, 21, 30]. This practice gives the user no knowledge going into the program. Some programs, such as, Offtonic Microtonal Synthesizer, and Scale Workshop were simple programs that did not need documentation [24, 25]. Very few apps, such as, Sound Modeler, and Melodyne 5, provided in-app explanations. These explanations can make the app usable without a tutorial [22, 31]. A few apps, such as, Blue, Melodyne 5, and Tune Smithy, provided extensive tutorials both in manual and video form [26, 29, 31]. To make an application understandable it should have some sort of outside tutorials or in-app information.

Some applications showcased what tools to give musicians to help them work well with other ATS. A few apps, such as Hex and Offtonic Microtonal Synthesizer, had unique GUI elements to visualize ATS [24]. Hex has a complex looking hexagonal lattice structure with lines drawn to show where the notes for each pitch are. This design is interesting, but it does not scale well, as systems become more complicated, the visualization becomes hard to understand [27].

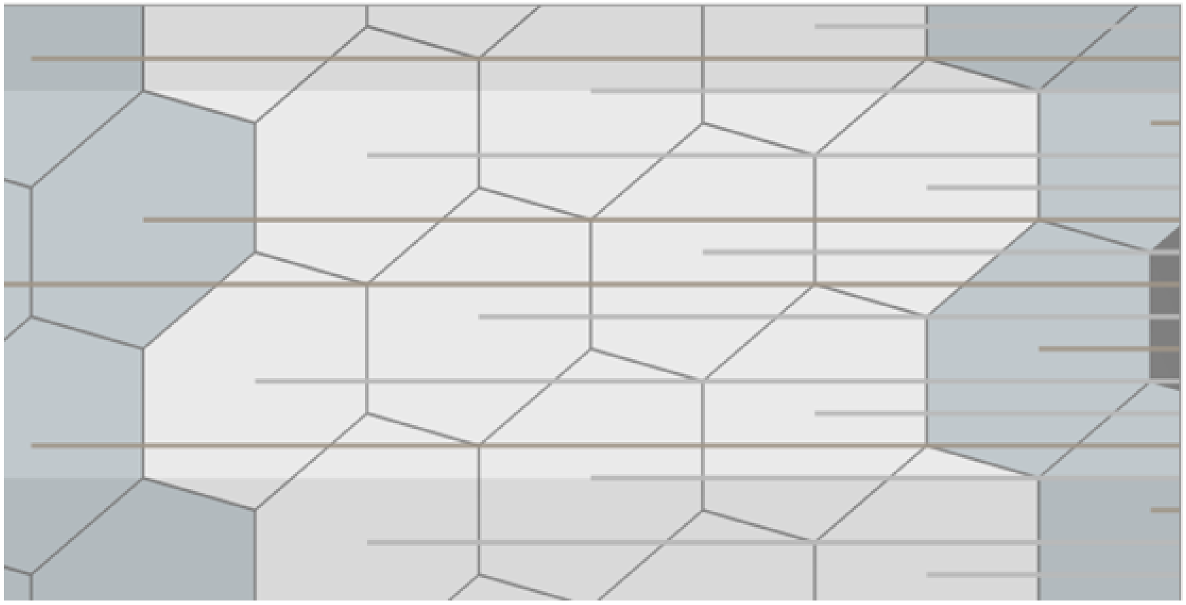


Figure 15 Hex Graphical Tuning Model [27]

From looking at a visualization we should be able to understand the ATS, what the notes should be called, and the frequencies of those notes. A few programs, such as Schismata, Melodyne 5, and Scale workshop provided good visualization that scaled well with ATS. They all provided a vertical modal that clearly displayed where the notes are. Melodyne 5 took the model a step further by allowing users to select which scales to view [31]. Schismata has a vertical bar that follows your cursor to show you which lines are which notes [28]. Scale workshop allows users to color code notes to create their own scales [25]. Music programs with different ATS should have some visualization to help users understand the system.

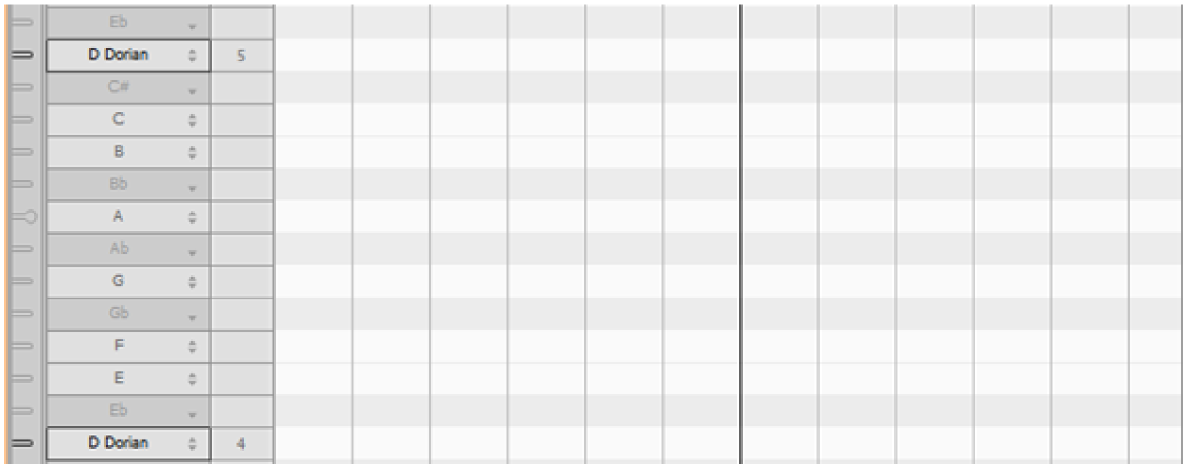


Figure 16 Melodyne 5 feature that can display different scales within a system. D Dorian is currently displayed [31].

Visualizing ATS is important, but first, there must be ATS to visualize. Many programs, such as Hex, Melodyne 5, Schismata, Scala, Tune Smithy, Offtonic Microtonal Synthesizer, and Scale workshop, offered preset tuning systems [20, 24, 25, 27-29, 31]. Presets can give musicians some systems to play around with, rather than just creating their own or looking for systems. Preset ATS should have a diverse background of both familiar and unfamiliar ATS, users should be able to understand what cultures ATS come from or what music they are used to create. While people can just research ATS, having the names and experimentation in one place may be valuable. Only Melodyne 5, from what we looked at, provided information on the ATS it represents. Each ATS has one sentence that provides little information, for example, one reads “Gamelan slendro from Ranchaiyuh, distr. Tanggerang, Batavia. 1/1=282.5 Hz (slendro2.scl)” [31]. Quite a few apps had system editing tools but only Schismata, Scale Workshop, and Scala provided tools we could use to create an interesting tuning system. Scale workshop provided an easy-to-use interface where we could simply type in numbers to create a tuning system [9]. Scala provided powerful creation tools through a command line interface and the ability to save ATS [20]. Schismata used their visual model to give the user a builder that handles multiple types of tuning systems [28]. ATS creation tools should provide an easy-to-use user interface.

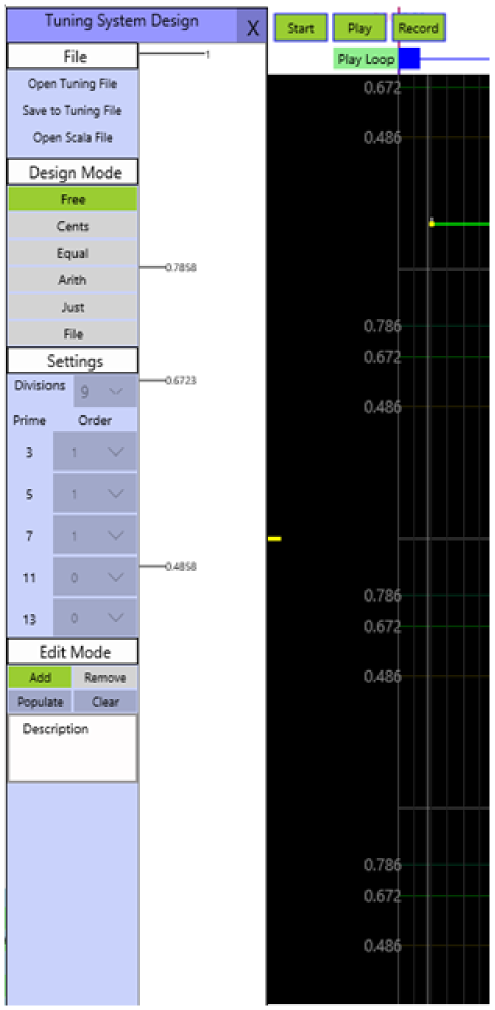


Figure 17 Schismata feature for creating and visualizing tuning systems [28].

While these tools help musicians build and compose music, they only allow one system to be used per song. A big part of exploring ATS is learning to create your own.

3 Methodology

3.1 Software Development Approach

When creating any piece of software, it is essential that the proper methodology is used. Using the wrong system or methodology can lead to setbacks, bugs in the code, and a lack of organization. In this section, we will compare the benefits and downsides of the Waterfall Methodology and Agile Methodology (Scrum), then we will select the best fitting methodology for us to use.

3.1.1 Waterfall

The Waterfall Method is a “linear-sequential life cycle model” which was the first type of software development life cycle approach that was ever used [32]. Essentially, this methodology consists of six stages, each stage must be completed before the next begins. In-order, the stages are requirements analysis, system design, implementation, integration and testing, deployment, and maintenance.

- The requirements analysis phase consists of gathering all the possible features and functionality for the piece of software. This can be done through many different methods such as interviews, focus groups, or surveys.
- The system design phase is where you start brainstorming and discussing architecture aspects the system. Lo-fi (low technology) prototypes and defining system hardware/software are both crucial aspects of this phase.
- The implementation phase takes all the requirements and design aspects of the first two phases and brings them together into small programs called “units” to make sure each of these aspects works as intended.

- The integrations and testing phase takes all the units from the previous phase and combines them into one system. This system is now tested for any bugs or problems that arise.
- The deployment phase is simply when the system is released to the public once all the bug testing has finished and the product is consumer ready.
- The maintenance phase involves creating "patches" that fix any issues that were only revealed once the consumers started using the system.

Overall, the Waterfall Methodology is effective when working with small projects whose requirements are well understood. Since this method is simple and rigid, it works with projects that change little throughout the development process [32].

On the other hand, there are numerous downsides to the Waterfall Approach. First, the development team will only have a working program very late in the life cycle. This is a major problem because it does not allow the team to learn and change their design as they code. It also limits the programmer's ability to spot potential bottlenecks or other integration problems early. Second, the rigidness of the Waterfall Approach is limiting for larger projects, especially ones that involve object-oriented coding, and projects whose requirements and design are expected to change greatly throughout the project's life cycle. Adjusting the scope of the project while in the middle of a cycle is often close to impossible [33]. Third, it is incredibly hard to measure your progress while in the middle of any given stage. This is a major problem as it can be impossible to estimate when you will finish the step you are on. Ultimately, the Waterfall Approach is fine for small programs but for anything with a bigger scope, the problems outweigh the benefits.

3.1.2 Agile Methodology

The Agile Methodology is a software development approach that uses iterative releases while emphasizing adaptability and customer satisfaction. The Agile Methodology believes that all projects are different and need to be handled as such [34]. In Agile, teams work in “sprints”, these are a set window of time where, at the end of that window, an iteration of the project must be completed [35]. Each sprint has its own goals, requirements, and scope. The requirements take the form of “user stories” that help put the developer in the mind of the customer, so they can come up with all the features necessary to create a complete, satisfactory product. Each sprint will present new user stories, and a backlog of unfinished use cases from the previous sprint to be integrated into the current iteration.

3.1.3 Scrum

Scrum is a subset framework of the Agile methodology and is currently one of the most used methodologies for software development [36]. A Scrum team consists of the product owner, the development team, and the scrum master. The product owner supplies the requirements, the development team does the development, and the scrum master oversees planning scrum meetings and removing obstacles. A typical Scrum sprint will have four parts, sprint planning, daily scrum, sprint review, and finally sprint retrospective [37].

- Sprint planning is simply where the project team plans the upcoming sprint. This includes gathering new requirements, deciding what backlog requirements take priority, and ultimately what product will be delivered at the end of the sprint.

- Daily scrum is a quick meeting that happens at least five times a week. In this meeting, each group member will report what they have done since the last meeting and what they are planning to do for the next meeting. The scrum master is responsible for planning and moderating the daily scrums.
- Sprint review is a meeting that happens at the end of a sprint. In this meeting, the development team showcases all the new features added in the sprint to the product owner. Then the team compares the finished iteration to the sprint goals they established in sprint planning. They use the comparisons to start planning the next sprint.
- Sprint retrospective is the opportunity for the members of the team to discuss what went well and what did not go so well in the sprint. These problems should be addressed and fixed for the next sprint.

Scrum is the most used methodology for good reason - it is simple and incredibly adjustable for the needs of almost any project. It also allows teams to produce functional programs incredibly early in the development cycle. This allows for the development team to see what works and what does not in real-time, instead of hoping everything works once they put it all together at the end. However, everything is not perfect with Scrum. In scrum there can be a noticeable lack of program documentation. This practice can lead to problems when transferring technology to new team members. Daily Scrums can also be unreasonable when there is a huge team working on one product. Agile Scrum teams should never have more than 10 members. Despite these factors, Scrum and the Agile Methodology are by far the best software methodology for most projects.

3.1.4 Conclusion

Ultimately, the Scrum methodology is by far the most suitable software methodology for us to use because of simplicity and flexibility. We will have seven week-long sprints with sprint planning before each and sprint review, and a retrospective after each release. During each sprint, we will have five daily scrum meetings.

3.2 Survey

We used a survey to gauge current musician’s knowledge on temperaments/tuning systems and their willingness to experiment and try out new systems. The survey also asks for their perceived level of musician ship (from Beginner to Professional) and their current level of music theory knowledge (from 1 to 5). We correlated the knowledge of temperaments with the musician’s levels to make conclusions on how current day musicians interact with temperaments, if at all. Furthermore, we added optional questions to allow surveyors to give their input of potential features in our plugin. This included questions about alternate keyboard layouts and sound synthesis. We used the results of this section to determine what features to add and what features to discard. The survey was sent only to musicians and their responses are all anonymous. View the survey questions in the appendix A.

4 Software Development Environment

A software development environment are the tools software engineers use to make software. To complete this project, we will use project management software, an integrated development environment, and digital audio workstations. Project management software allows us to organize project tasks and track project progress, manage multiple people working on one program, and store the program. Integrated development environments give engineers an environment to write, test, and develop the software. Digital audio workstations are the platform we are building a VST for so we will need a few DAWs to test the software in.

4.1 Project Management Software

Project management software allows more control and flexibility when creating a project. This kind of software is important for any software development methodology. First, we need software to plan and track our progress. Jira is Atlassian software that is used in the professional software engineering field. It allows users to carefully plan out and track the progress of the entire project. Users can create epics for large portions of the project, and issues for specific development steps. Users can view the steps we completed in each iteration so we can review our work [38]. We chose Jira because we are both familiar with using it to manage software projects. Next, we need software to manage the project code. Git is a version control software designed for projects of all sizes. It allows the management of different builds of the project simultaneously, so different members can work on separate parts of the program. Users can merge the program so other users can smoothly integrate their parts with the rest of the program. Anyone who has the URL can clone the repository so they can work on the program's source code [39]. Then, to use git users need somewhere to store the code so we can both easily access it. Bit Bucket is another Atlassian software, it is used to create a private code repository. It provides URLs used to access different repositories or branches. We chose this software because it works well with Jira so we can connect issues to branches [40]. Source Tree is another Atlassian software, it is used as a git client. It provides an easy-to-use graphical user interface to git, rather than using the git command line interface. It can display the version history for all the project branches so you can easily manage each one. It provides an interface to show all the differences between the current and previous versions of software [41]. Overall, we chose to use Jira 8.15 and Bit Bucket 7.1 because they are familiar to the team and work well together.

4.2 Integrated Development Environment

To write, build, and debug the software project teams need an Integrated development environment. To create a VST we need a source development kit (SDK) from Steinberg, and an IDE. For a framework we used JUCE 6.0.4, which is set up to build VSTs. JUCE provides VST graphical user interface (GUI) elements so we can give the user controls and the ability to code the project in the programming language C++. C++ is a common language for VSTs and the authors have experience programming in this language. JUCE by itself does not work as a compiler and builder, therefor we used Visual Studio 2019 6.9 for editing, writing, and compiling our code.

4.4 Digital Audio Workstations

To test the software, we needed to check a variety of digital audio workstations to make sure the VST works. Each DAW implements plugins slightly differently, so the more platforms we test on, the more likely the VST is to work on additional platforms. We will be using FL Studio, Ableton, Reaper, and Garage Band. These are among the leading digital audio workstations which many professionals use [8].

4.5 Survey

We used WPI's Qualtrics survey tool for creating and distributing the survey. We chose to use this survey tool for two reasons. First, it is free to use for WPI students. Second, by using this tool we ensure that the data is owned by WPI.

5 Software Requirements

5.1 Strategy

The first step for requirements gathering was brainstorming ideas. Then we continued with our survey and literature review to gather more requirements. Finally, user stories, use case diagrams, and lo-fi UI mockups followed shortly once the vision for what the plugin should look like was well-defined.

5.2 Brainstorming

For brainstorming, we set a time limit and came up with as many ideas as possible for potential features and UI designs, we wanted out plugin to have. After the time limit, we went through all our ideas to determine what we wanted to keep and what we wanted to discard.

During brainstorming we concluded that our plugin should have three different aspects to it. First and most importantly, there will be a preset bar where users can select from a list of pre-programmed systems. These systems will be fully implemented and ready to go, the user should immediately be able to start recording music using the new system. We will have full midi functionality for most midi devices. Therefore, this aspect of the plugin should be as plug-and-play as possible requiring very little on the user's end. We are emphasizing the accessibility in this part of the plugin. We think that while musicians are experimenting and trying out new things, they do not want to be slowed down by confusing user interfaces or complex set-ups. Some of the preset systems will have system variations the user can choose from by selecting the sub-system.

Second, we want the plugin to have an educational aspect where the user can learn more about whatever preset system they choose to use. This will manifest in a non-obtrusive info button somewhere in the top right corner of the user interface. If the user decides they want to learn more about the system they are working in, or if they want to learn how to use the system, a short, yet, extensive set of information will be available in the information section. This will include some history on the system, how the system works, and some small music theory tips towards using the system. We want this section to be completely optional, we do not expect every user to desire this functionality. However, it is important that people that want to learn how to use a system, should be given proper information.

Third, we want the plugin to have a system-creation aspect to it. For musicians who want more than half a dozen presets, they will most likely turn to system creation to either replicate another system they know about or create their own tuning system. This aspect will take form in various "builders" for some system. The freeform builder is the most open of all builders, it allows the user to directly map pitches to MIDI devices and create their own system with any pitches they want. Another builder will be the "Equal Temperament" builder, where the user can enter how many notes between an octave they want, the size of each octave, and the builder will create an equal temperament system for them. We will give the user more control in this section, so the user will not be restricted by any rules.

5.3 Literature Review

One form of requirements gathering was the literature review in section 2.2, where we looked at previous programs and VSTs. To make the program more accessible, the download or install process should be as simple as possible. The program should not trigger the user's ani-virus, since this action could be a red flag to users. To help users understand how to operate the program, some documentation should be provided to explain how the program works. To help users understand other tuning systems, the app should provide multiple tuning systems and brief explanations of each. To help users understand the system layout, the app should have some sort of visual representation of each system. The app should give the user tools to create or alter preset systems, so they can make their own unique system.

5.4 Functional and Non-functional Requirements

Functional Requirements:

- The plugin should allow users to easily use any of the ATS presets provided to them.
- The plugin should allow users to seamlessly record and create music in any DAW that supports VST plugins.
- The plugin should provide users with concise information on each ATS .
- The plugin should allow the user to create their own unique ATS .
- The plugin should allow users to control sound design and timbre.
- The plugin should save the user's "patches" so they can access them in other projects.

Non-functional Requirements:

- The plugin should be simple to use and understand.
- The plugin should work well with other VST plugins. Specifically, audio effects VSTs, to alter the timbre of the plugin.
- With the proper hardware, there should be less than 100 milliseconds of latency when playing through the plugin.
- The plugin should load in under 5 seconds on most modern computers.

5.5 Epics and User Stories

5.4.1 Initial Epics and User stories

The Initial epics and user stories were established before we started programming the plugin.

Epic #1: Have presets for users to try out established presets.

- As a musician, I want to select a preset ATS so that I can use that ATS in my music in a DAW.
- As a musician, I want to use equal temperament to create music.
- As a musician, I want to use Just Intonation to create music.
- As a musician, I want to use mean tone temperament to create music.
- As a musician, I want to use Well-Temperament to create music.
- As a musician, I want to use Maqam to create music.
- As a musician, I want to use Gamelan to create music.
- As a musician, I want to be able to record my music in any ATS I select to integrate the new ATS into my music.

- As a musician, I want to explore different variations of the preset ATS to further experiment with ATS.

Epic #2: Provide Information to the users so they know what a system is and how it works

- As a musician, I want to know the origin and brief history of the ATS I am using so I have some background knowledge before using a ATS.
- As a musician, I want to know how different aspects of the ATS function, and some basic music theory (if possible), so I can understand my musical options when using a ATS.
- As a musician, I do not want any extra information to be obtrusive to the main UI so that I can ignore the extra info if I want.

Epic #3: Allow users to create their own systems

- As a musician, I want to create my own musical ATS from scratch to experiment with new ATS.
- As a musician, I want to assign certain frequencies to certain keys to create a completely new ATS.
- As a musician, I want to create an equal temperament system that is not 12 notes to experiment with new ATS.
- As a user of the plugin, I want to be able to save my ATS as files so I can use them as many times as I need without recreating them.
- As a musician, I want to "deconstruct" the preset ATS to see how they were made (only applies to presets made with a builder).

- As a musician, I want to be able to build ATS using any preset ATS as a starting point, so I am not going in completely blind when making my own ATS.

Epic #4: Midi integration

- As a musician, I want to use my MIDI controller/keyboard to play tones from the current ATS to easily try out new ATS.
- As a musician, I want to use my computer keyboard to play tones from the current ATS to easily try out new ATS without spending money on a MIDI device.
- As a musician, I want to choose the mapping of my MIDI keyboard so I can make an easy-to-use layout.
- As a musician, I want to be able to use an on-screen keyboard to play tones from the current ATS to easily try out new ATS without buying a MIDI device.
- As a musician, I want to use my own samples for whatever ATS I choose so I can incorporate the ATS into my music and have the timbre fit.
- As a musician, I want to have a simple synthesizer or be able to choose a sample preset sound, so I can use the ATS in my music without needing to find good samples.
- As a musician, I want to be able to use my launchpad and other pad-based midi devices with the program so I can easily try out new ATS without having to buy a MIDI keyboard.
- As a musician, I want to know the exact frequency I am playing so I can quantify each tone.
- As a musician, I want the keys on the virtual keyboard to light up when I play them to give me clarity as to what note I am playing.

- As a musician, I want to be able to change the keyboard layout to a non-formatted keyboard, so the ATS layout does not contradict the keyboard layout on screen.
- As a musician, I want to have the default MIDI mapping format be consistent for keyboard layout, so the mapping stays the same from octave to octave.

Epic #5: Have an informative, but concise, user manual.

- As a user of the plugin, I want to know what buttons inside the plugin do so I know how to operate the plugin.
- As a user of the plugin, I want to know how I can record with the plugin.
- As a user of the plugin, I want to know how I can do midi mapping so I can map systems to my midi device.
- As a user of the plugin, I want to know how ATS creation work so I can create new systems.
- As a user of the plugin, I want to know how to use preset ATS so I can try out the preset systems.
- As a user of the plugin, I want to easily find documentation and search for specific parts of the program (documentation should not just be one big manual).
- As a user of the plugin, I want to have the documentation in pdf text and picture form so I can download it for easy access and skim through it quickly.

Epic #6: Safe, reliable, trustworthy distribution

- As a potential user, I want to know this program is safe (does not trigger anti-virus software) so I can trust using the software.

- As a developer, I want to know that we are in full compliance with any license agreements before we release the software, so we are not unknowingly breaking any rules.
- As a potential user, I want a one-click-installer executable that installs the software with all its dependencies to be readily used in my DAW.
- As a potential user, I want to know what the acceptable use policy for a program is before I start using it, so I am not unknowingly breaking any rule.
- As a potential user, I want to be able to see or contact the program's developers if I have a problem using the program.

5.4.2 Final Epics and User stories

The final epics and user stories were established after programming the plugin. Some of the initial ones were taken out and new stories were added.

Epic #1: Have presets for users to try out established presets.

- As a musician, I want to select a preset ATS so that I can use that ATS in my music in a DAW.
- As a musician, I want to use equal temperament to create music.
- As a musician, I want to use Just Intonation to create music.
- As a musician, I want to use mean tone temperament to create music.
- As a musician, I want to use Well-Temperament to create music.
- As a musician, I want to use Maqam to create music.
- As a musician, I want to use Gamelan to create music.
- As a musician, I want to use Genesis to create music.

- As a musician, I want to be able to record my music in any ATS I select to integrate the new ATS into my music.
- As a musician, I want to explore different variations of the preset ATS to further experiment with ATS.
- As a musician, I want to control the sustain of the synth, so I can have a bigger selection of sounds to use.
- As a musician, I want to control the decay of the synth, so I can have a bigger selection of sounds to use.
- As a musician, I want to control the attack of the synth, so I can have a bigger selection of sounds to use.
- As a musician, I want to control the release of the synth, so I can have a bigger selection of sounds to use.
- As a musician, I want to control the wave form of the synth, so I can have a bigger selection of sounds to use.
- As a musician, I want to control the wave form of the synth, so I can have a bigger selection of sounds to use.
- As a musician, I want to control the filter cutoff of the synth, so I can have a bigger selection of sounds to use.
- As a musician, I want to control the filter resonance of the synth, so I can have a bigger selection of sounds to use.

Epic #2: Provide Information to the users so they know what a system is and how it works.

- As a musician, I want to know the origin and brief history of the ATS I am using so I have some background knowledge before using a ATS.

- As a musician, I want to know how different aspects of the ATS function, and some basic music theory (if possible), so I can understand my musical options when using an ATS.
- As a musician, I do not want any extra information to be obtrusive to the main UI so that I can ignore the extra info if I want.

Epic #3: Allow users to create their own systems.

- As a musician, I want to create my own musical ATS from scratch to experiment with new ATS.
- As a musician, I want to create an equal temperament system that is not 12 notes to experiment with new ATS.
- As a user of the plugin, I want to be able to save my ATS to “patches” so I can use them as many times as I need without recreating them.
- As a musician, I want to be able to build ATS using any preset as a starting point, so I am not going in completely blind when making my own ATS.

Epic #4: Midi integration

- As a musician, I want to use my MIDI controller/keyboard to play tones from the current ATS to easily try out new ATS.
- As a musician, I want to use my computer keyboard to play tones from the current ATS to easily try out new systems without spending money on a midi device.
- As a musician, I want to be able to use an on-screen keyboard to play tones from the current ATS to easily try out new ATS without buying a midi device.

- As a musician, I want to be able to use my launchpad and other pad-based midi devices with the program so I can easily try out new ATS without having to buy a MIDI keyboard.
- As a musician, I want to know the exact frequency I am playing so I can quantify each tone.
- As a musician, I want the keys on the virtual keyboard to light up when I play them to give me clarity as to what note I am playing.
- As a musician, I want to have the default MIDI mapping format be consistent for keyboard layout, so the mapping stays the same from octave to octave.

Epic #5: Have an informative, but concise, user manual

- As a user of the plugin, I want to know what buttons inside the plugin do so I know how to operate the plugin.
- As a user of the plugin, I want to know how I can record with the plugin.
- As a user of the plugin, I want to know how I can do midi mapping so I can map systems to my midi device.
- As a user of the plugin, I want to know how ATS creation work so I can create new systems.
- As a user of the plugin, I want to know how to use preset ATS so I can try out the preset ATS.
- As a user of the plugin, I want to easily find documentation and search for specific parts of the program (documentation should not just be one big manual).
- As a user of the plugin, I want to have the documentation in pdf text and picture form so I can download it for easy access and skim through it quickly.

Epic #6: Safe, reliable, trustworthy distribution

- As a potential user, I want to know this program is safe (does not trigger anti-virus software) so I can trust using the software.
- As a developer, I want to know that we are in full compliance with any license agreements before we release the software, so we are not unknowingly breaking any rules
- As a potential user, I want a one-click-installer executable that installs the software with all its dependencies to be readily used in my DAW.
- As a potential user, I want to know what the acceptable use policy for a program is before I start using it, so I am not unknowingly breaking any rule.
- As a potential user, I want to be able to see or contact the program's developers if I have a problem using the program.

5.6 Use Cases

5.4.1 Initial Design Use Cases

The Initial use cases were established before we started programming the plugin.

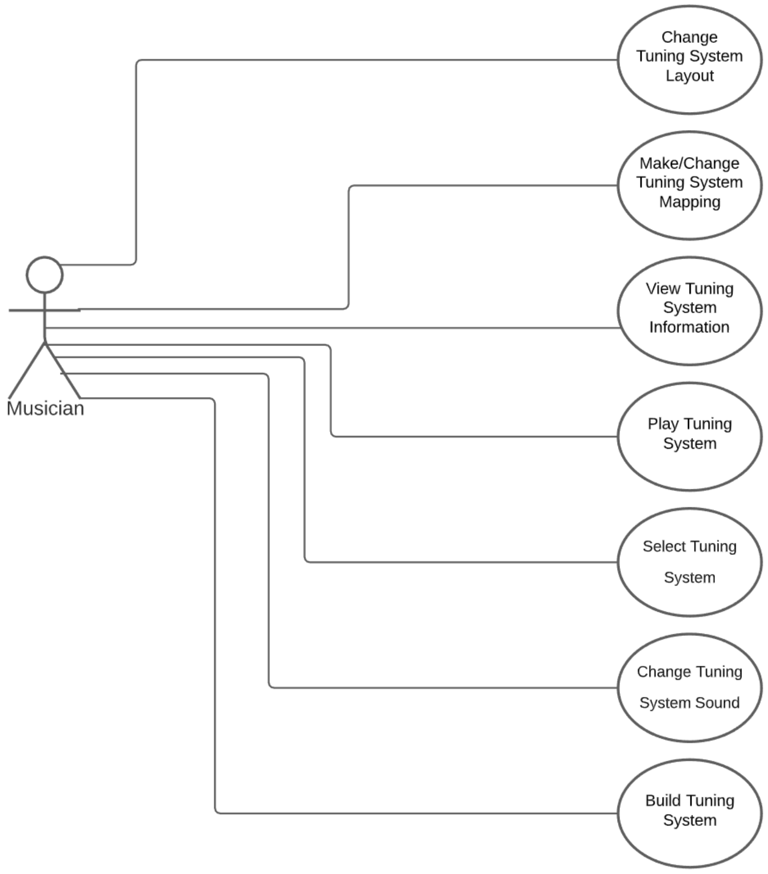

Change Tuning System Layout
Make/Change Tuning System Mapping
View Tuning System Information
Play Tuning System
Select Tuning System
Change Tuning System Sound
Build Tuning System
Musician

Figure 18 initial UML use case diagram

Unique name: Select Tuning System

Participating actors: Musician

Entry condition:

- Musician has none or any previous ATS selected.

Exit conditions:

- Musician closes the VST

Flow of events:

1. Musician selects an ATS, either a preset or user created ATS

Special requirements: None

Unique name: Play Tuning System

Participating actors: Musician

Entry conditions:

- Musician selects an ATS, include (Select Tuning System)

Exit conditions:

- Musician closes the VST
- Musician selects a different ATS, include (Select Tuning System)

Flow of events:

1. (optional) Musician can change how the system is played with any of the following options.
    - Musician selects a sound, include (Select/Change Tuning System Sound)
    - Musician selects a key layout, include (Change Tuning System Layout)
    - Musician maps/remaps they layout, include (Make/Change Tuning System Mapping)
    - Musician changes ATS parameters (if possible), include (Build Tuning System)
2. Play a tone from one of the following options
    - Play an enabled key/button on a connected MIDI controller
    - Play an enabled key on a connected computer keyboard
    - Click an enabled key on the on-screen model

Special requirements: None

Unique name: View Tuning System Information

Participating actors: Musician

Entry conditions:

- Musician selects an ATS, include (Select Tuning System)

Exit conditions:

- Musician closes the ATS information
- Musician selects a different ATS, include (Select Tuning System)

Flow of events:

1. Musician requests any of the following information
    - Musician requests to view ATS history/creation
    - Musician requests to view ATS functionality or music theory
2. Musician can save the ATS settings (if applicable), include (Build Tuning System)

Special requirements: None

Unique name: Make/Change Tuning System Mapping

Participating actors: Musician

Entry conditions:

- Musician selects an ATS, include (Select Tuning System)

Exit conditions:

- Musician closes the VST
- Musician stops remapping
- Automatic mapping finished
- Musician selects a different ATS, include (Select Tuning System)

Flow of events:

1. Musician can request to have any ATS auto-mapped, include (Change Tuning System Layout)
2. Musician selects a key on the model, include (Change Tuning System Layout)
3. (optional) Musician assigns a tone
4. Musician presses a button on either of the following devices
    a. Connected MIDI controller
    b. Connected Computer keyboard
5. Musician can save the ATS settings, include (Build Tuning System)

Special requirements: The state of the automapping depends on the current ATS layout.

Unique name: Build Tuning System

Participating actors: Musician

Entry conditions:

- Musician selects to build an ATS, include (Select Tuning System)

Exit conditions:

- Musician cancels building the ATS
- Musician selects a different preset ATS, include (Select Tuning System)
- Musician finishes building the ATS
- Musician closes the VST

Flow of events:

1. Musician can build from either stating point
    - Equal Temperament Builder

        - Free Builder, (optional) preset, include (Select Tuning System)
        - Customizing ATS options (if applicable)
2. Musician can adjust any of the ATS parameters, depends on starting point
3. Musician can add information to the ATS, include (View Tuning System Information)
4. Musician can save the ATS
5. Musician can play the ATS, include (Play Tuning System)

Special requirements: Building an ATS should have a sub controller for each building starting point.

Unique name: Select/Change Tuning System Sound
Participating actors: Musician

Entry conditions:

- Musician selects an ATS, include (Select Tuning System)

Exit conditions:

- Musician finishes changing the ATS sounds
- Musician closes the VST

Flow of events:

1. Musician can select a sound setup from any of the following options
    - Select preset sample
    - Import sample
    - Simple synthesizer
2. Musician can save the current sample set, includes (Build Tuning System)

Special requirements: None

Unique name: Change Tuning System Layout

Participating actors: Musician

Entry conditions:

- Musician selects an ATS, include (Select Tuning System)

Exit conditions:

- Musician finishes ATS layout
- Musician closes the VST

Flow of events:

1. Musician can select from any of the layout setups
    - (default) Piano layout
    - Minimal formatting
    - Automatic formatting, include (Build Tuning System)
2. Musician can save the ATS settings, include (Build Tuning System)

Special requirements: Automatic formatting only works for equal temperament ATS.

5.4.2 Final Design Use Cases

The Final use cases were established after we had finished programming the plugin.

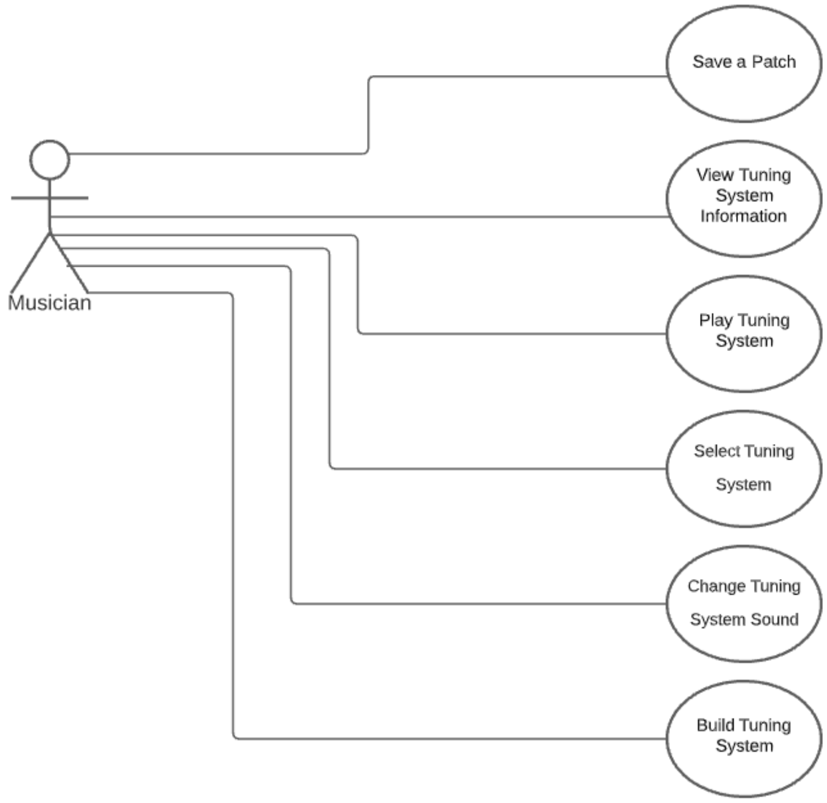


Figure 19 Final UML use case diagram

Unique name: Select Tuning System

Participating actors: Musician

Entry condition:

- Musician has none or any previous system selected

Exit conditions:

- Musician closes the VST

Flow of events:

1. Musician selects an ATS
2. Musician selects user-created patch

Special requirements: None

Unique name: Play Tuning System

Participating actors: Musician

Entry conditions:

- Musician selects an ATS, include (Select Tuning System)

Exit conditions:

- Musician closes the VST
- Musician selects a different ATS, include (Select Tuning System)

Flow of events:

1. Play a tone from one of the following options
    - Play an enabled key/button on a connected MIDI controller
    - Play an enabled key on a connected computer keyboard

Special requirements: None

Unique name: View Tuning System Information

Participating actors: Musician

Entry conditions:

- Musician selects an ATS, include (Select Tuning System)

Exit conditions:

- Musician closes the ATS information using the info button

Flow of events:

1. Musician clicks on the information button
2. Musician views the information for the ATS that is selected

Special requirements: None

Unique name: Build Tuning System

Participating actors: Musician

Entry conditions:

- Musician Selects Equal Temperament as their ATS

Exit conditions:

- Musician selects a different preset ATS, include (Select Tuning System)
- Musician finishes building the ATS
- Musician closes the VST

Flow of events:

1. Musician can adjust any of the system parameters through three different sliders
2. Musician clicks the build button

Special requirements: None

Unique name: Select/Change Tuning System Sound
Participating actors: Musician

Entry conditions:

- Musician selects an ATS, include (Select Tuning System)

Exit conditions:

- Musician finishes changing the ATS sounds.
- Musician closes the VST.

Flow of events:

1. Musician controls the ADSR envelope of the synth through sliders
2. Musician selects the waveform of the synth through a drop-down menu
3. Musicians controls the filter cutoff and resonance of the synth though sliders
4. Musician can save their sound settings at as a patch in the DAW (extends)

Special requirements: None

Unique name: Save patch
Participating actors: Musician

Entry conditions:

- Musician selects an ATS, include (Select Tuning System)

Exit conditions:

- Musician closes the VST

Flow of events:

1. Musician saves a patch of the plugin’s current settings to their daw

Special requirements: None

5.7 Interface Mockups

From our literature review, the most important takeaway was the fact that plugins almost always do not take up the whole screen on any given device. We decided that we needed to be incredibly conscious of space and make a program that fits to a quarter or third of most computer screens while also not feeling over-cramped. Furthermore, most plugins with some sort of midi aspect have a virtual keyboard spanning across the bottom of the program for users to click on if they do not own a physical midi device. Since our program works closely with midi the virtual keyboard is essential to include. Furthermore, many DAWs, such as Reaper, take handle presets in their own UI, however these presets only hold information that the user themselves can change, therefore we still need to include an additional menu for selecting the system.

We used Balsamiq 4.2.1 to create the UI mockups. Our first mockup was too big and had too much unused space, so we shrunk it down to become smaller. We wanted the layout of the tools to make sense and flow nicely.

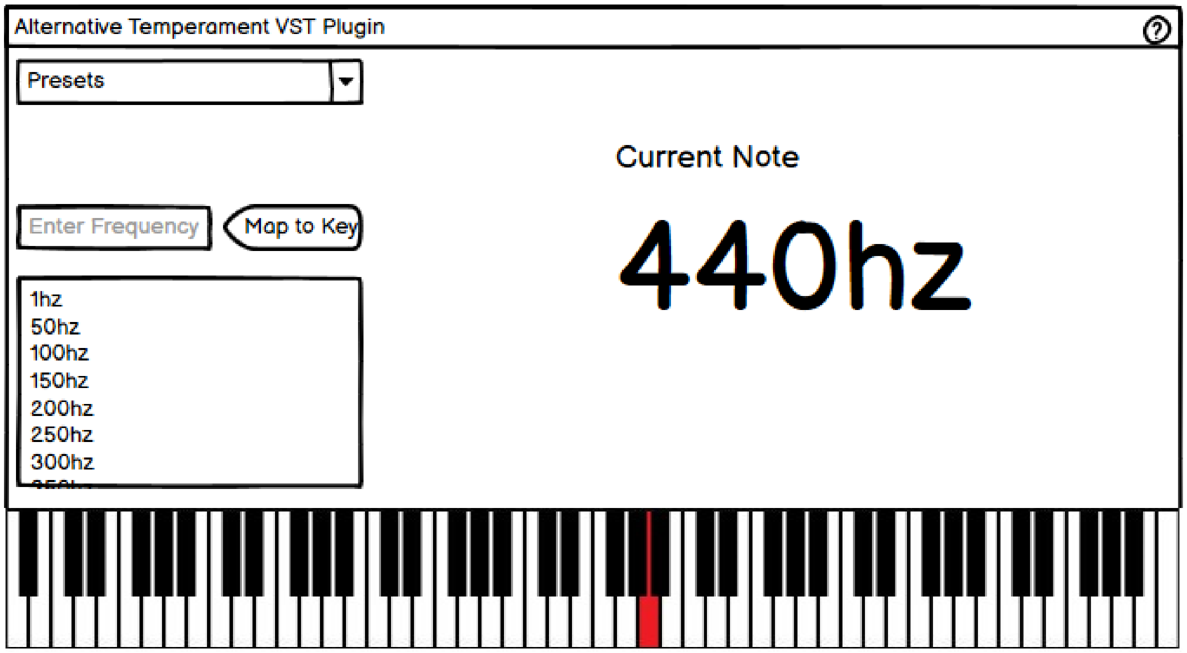


Figure 20 Large Version, User Interface Mockup

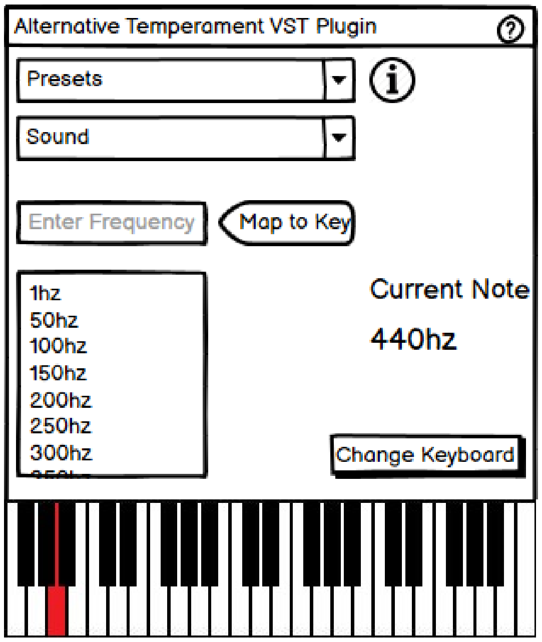


Figure 21 Compact Version, User Interface Mockup

6 Design

6.1 Existing Software Frameworks or Architectures

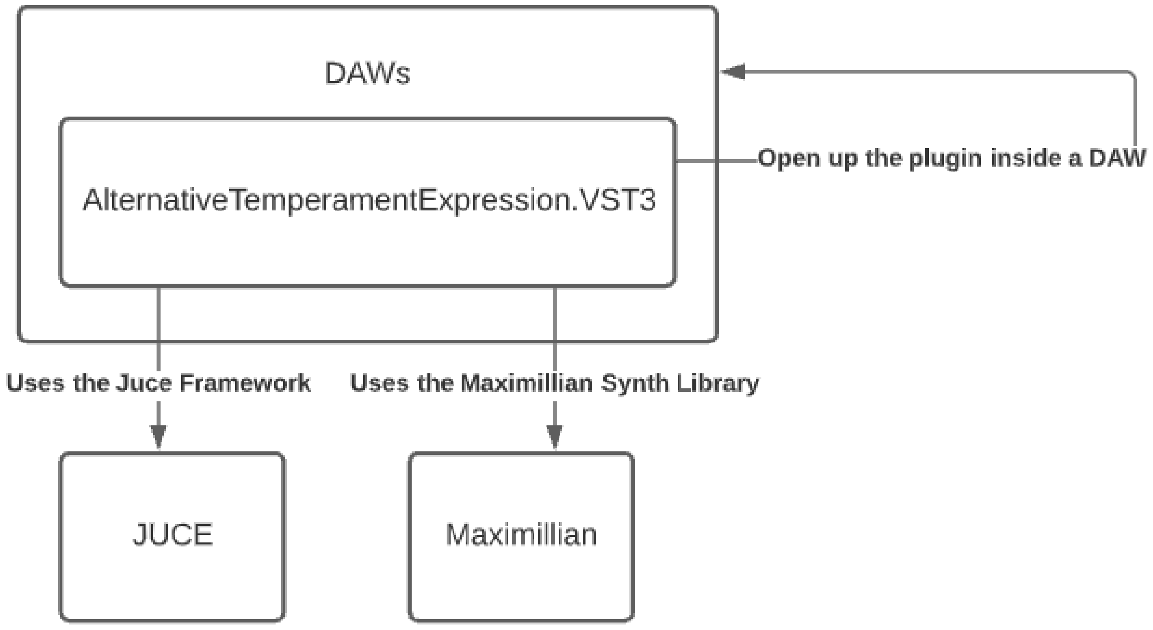

DAWs
AlternativeTemperamentExpression.VST3
Open up the plugin inside a DAW
Uses the Juce Framework
Uses the Maximillian Synth Library
JUCE
Maximillian

Figure 22 Software Environment Diagram

6.2 User Interface Design

We wanted our final UI design to be simple and easy to understand. While it is not perfect, we made it as professional-looking to the best of our abilities.

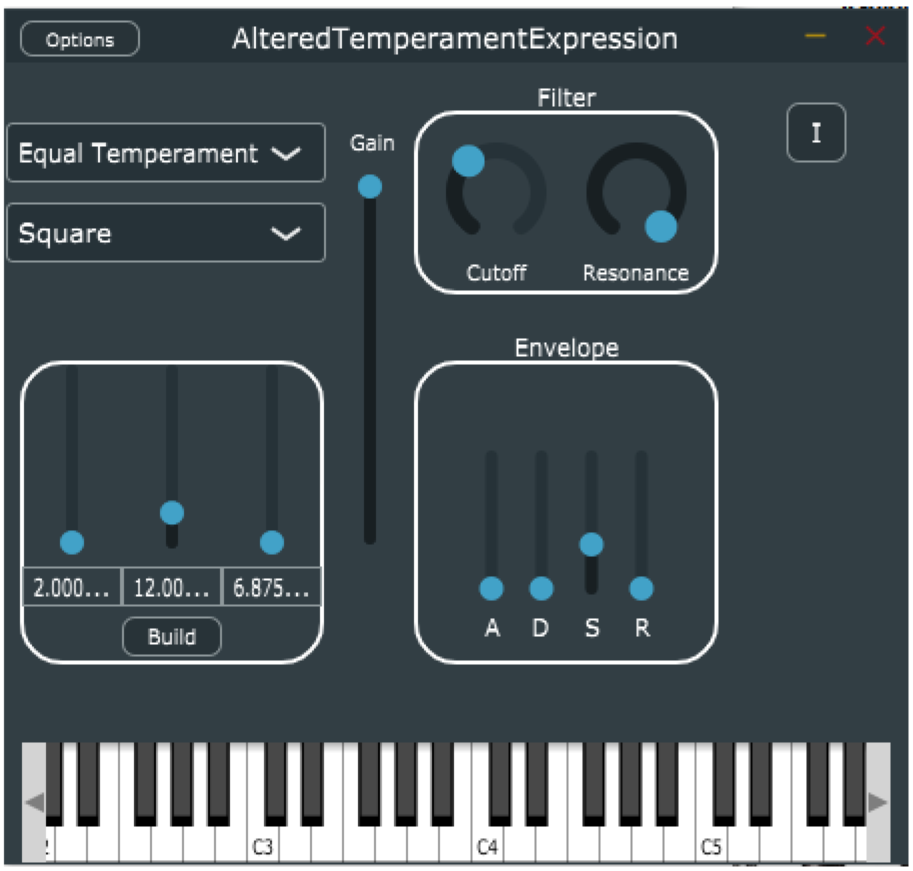
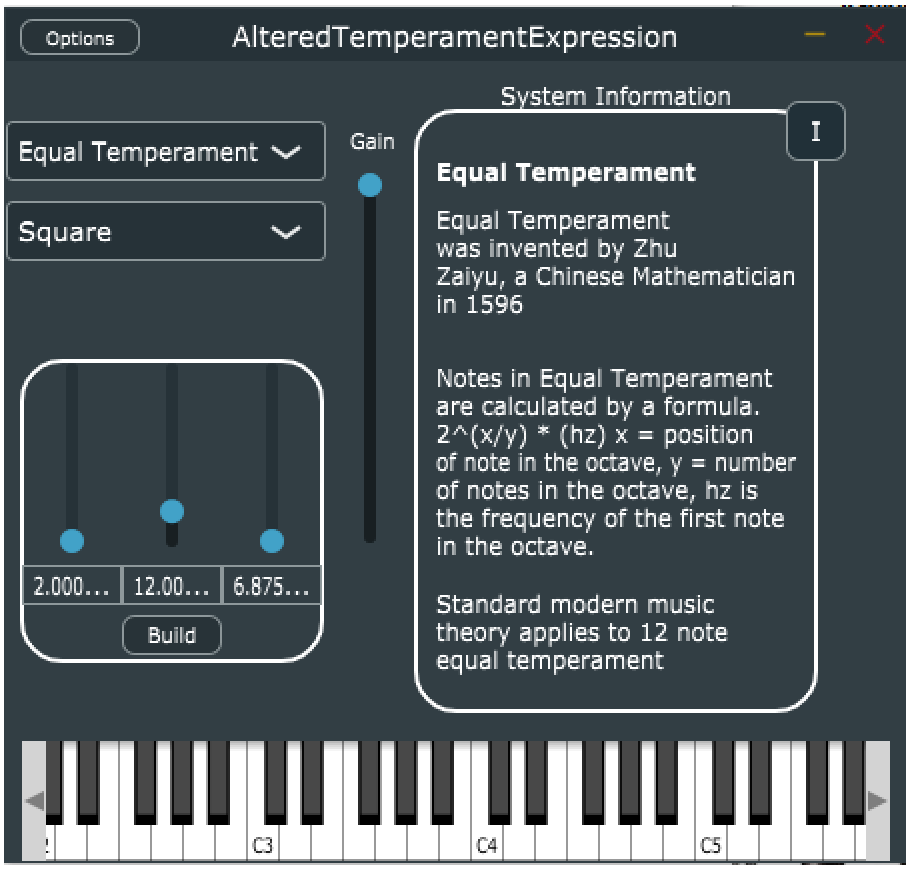


Figure 23 Final UI design



Equal Temperament

Equal Temperament was invented by Zhu Zaiyu, a Chinese Mathematician in 1596

Notes in Equal Temperament are calculated by a formula. 2^(x/y) * (hz) x = position of note in the octave, y = number of notes in the octave, hz is the frequency of the first note in the octave.

Standard modern music theory applies to 12 note equal temperament

Figure 24 Figure 20 Final UI design with info section

6.3 Models

We will use a few different models to display ATS to the user. The main model is a simple piano keyboard that shows some range of multiple octaves. Keys will light up when the ATS is playing, and keys will be greyed out when they are not attached to any tone. This model will be the default view to give western users familiarity and match typical MIDI keyboards.

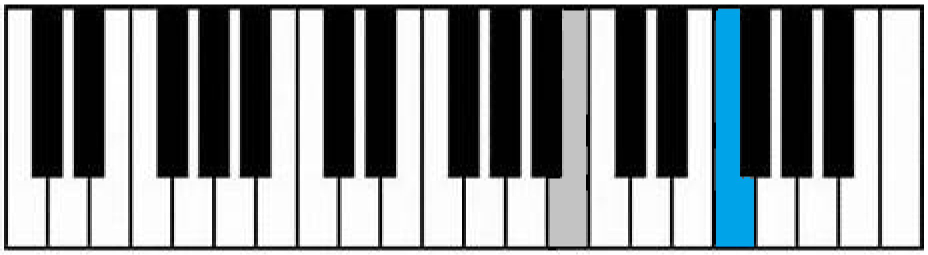


Figure 25 keyboard model. The blue key is currently being played. The grey key is not attached to a tone.

This model does not make musical sense for all ATS so we may offer other models that are more flexible. Constructing alternative models may require us to create specialized GIU objects. One idea for an alternative model is an unformatted keyboard with only one type of key. The following figure shows a possible model.

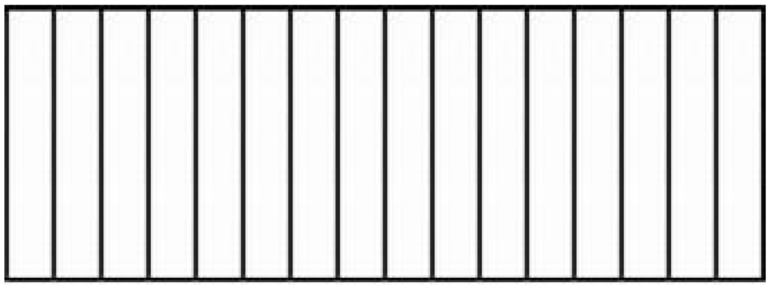


Figure 26 Unformatted keyboard model

Just Intonation involves a tuning setup that uses only whole number ratios, instead of tempered ratios, to create tones. This definition gives us a simple model, where we can use a list of ratios that cycle once per octave, to create a flexible model. We can use this model to generate tones for any size ATS or starting point frequency. As an example, we can use the following possible Just Intonation setup for 12-tones per octave.

Tone name	ratios	Frequency (in Hz)
A	1/1	440
A-sharp/B-flat	16/15	469.33333
B	9/8	495.00000
C	6/5	528.00000

C-sharp/D-flat	5/4	550.00000
D	4/3	586.66667
D-sharp/E-flat	17/12	623.33333
E	3/2	660.00000
F	8/5	704.00000
F-sharp/G-flat	5/3	733.33333
G	16/9	782.22222
G-sharp/A-flat	15/8	825.00000
A	2/1	880

Figure 27 4 12-tone Base 2 Just Intonation example (Repeat)

We can select a starting point, for example 440 Hz, then multiply it by each ratio to get that tone within an octave. For example, 440 * 9/8 = 495 Hz for B. To create tones outside the first octave, we can multiply or divide any number within the octave by some power of the base or octave size. For example, take the ratio 3/2 * 440 = 660 Hz, then 660 * 2^1 = 1320 Hz will be the E for the octave above the first octave. The 2^N is how many octaves above or below the first or center octave. To alter the ATS, the user can select the center frequency, the start index, and the end index. The center frequency is the frequency for the ratio 1/1. The start index is a negative integer that represents how many tones the user wants below the center. The end index is a positive integer that represents how many tones the user wants above the center. The start and end will be altered as needed to fit the system to MIDI or other messaging systems, but the program will keep the center frequency in the center of the tuning system. The bounds of the system should fall within human hearing range, that is approximately 20 Hz to 20 kHz [42]. We can warn the user that the ATS is beyond human hearing range. The ATS bound properties also apply to meantone temperament, well temperament, and equal temperament.

Meantone temperament is not one temperament system, there are many variations that allow musicians to use approximately 75% of the temperament. Meantone temperament can be modeled using the ratios between half-steps for a single octave. We can use this model to generate tones for any size ATS or starting point frequency. As an example, we can use 17th-centry Irregular Equal-beating mean-tone Temperament. This table does not include the optional corrections [9]. The method for calculating the frequencies is the same method as just intonation.

Tones	half-step ratios	Tones (in Hz)
A	1	440
A-sharp/B-flat	1.068168972	469.9943479
B	1.048233588	492.6638614
C	1.068232322	526.2794604
C-sharp/D-flat	1.048185725	551.6386178
D	1.066901626	588.5441383
D-sharp/E-flat	1.067951509	628.5366005
E	1.047817897	658.5918986
F	1.068314717	703.5834181
F-sharp/G-flat	1.048057146	737.3956294
G	1.066731285	786.6029874
G-sharp/A-flat	1.04835824	824.6417232
A	1.067130094	880

Figure 28 17th-centry Irregular Equal-beating mean-tone Temperament (Repeat)

Well temperament is not one ATS, there are many variations that allow musicians to use all keys within a tuning system. Well temperament can be modeled the same as Meantone temperament. One way to model any Well-Temperament system is to model the ratio between half-steps for a single octave. We can use this model to generate tones for any size ATS or starting point frequency. As an example, we can use Francesco Antonio Vallotti’s Early Eighteenth-Century Well Temperament [9]. The method for calculating the frequencies is the same method as just intonation.

Tones	half-step ratios	Tones (in Hz)
A	1	440
A-sharp/B-flat	1.063329484	467.8649728
B	1.053867457	493.0676691
C	1.065212939	525.222061
C-sharp/D-flat	1.056126862	554.7011272
D	1.060280738	588.1389207
D-sharp/E-flat	1.060667713	623.819964
E	1.056227518	658.8958121
F	1.065111427	701.7974588
F-sharp/G-flat	1.053867457	739.6015033
G	1.062271339	785.6574792
G-sharp/A-flat	1.059051447	832.0516903
A	1.0576266	880

Figure 29 Francesco Antonio Vallotti's Early Eighteenth-Century Well Temperament (Repeat)

Equal temperament has a different layout scheme than other temperaments because the builder can easily change the temperament's tones per octave, and octave size. Equal temperament can be defined by any size of consistent half-step spacing, not just 12 tones per octave. We chose a standard keyboard layout to make the program familiar to western musicians, and since many MIDI devices have keyboard formatting. Keyboard formatting refers to the grouping of black and white keys that clearly shows the major C scale. When moving to other forms of equal temperament, the major C scale will have less, or more tones per octave, and have a completely different formatting. Therefore, fitting other equal temperaments onto the same formatting as 12-tone equal temperament does not make sense musically. We think we need to conform to musical standards by providing a consistent formatting algorithm for any equal temperament system to the 12-tone layout. We want to keep the octave location consistent; this constraint means that ATS below 12-tones per octave will fit within each octave, and ATS above 12-tones per octave will take only as many octaves as necessary.

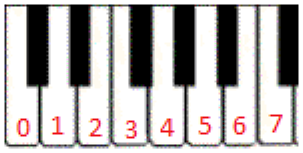


Figure 30 7-tone equal temperament layout

This figure shows a possible layout for 7-tone equal temperament that fits within an octave. The frequencies will depend on the rest of the builder parameters the user sets. This layout may be the default, but the user can always switch the layout to something more comfortable. The next figure shows a possible layout for 19-tone equal temperament. This temperament layout will fit within two octave spaces.

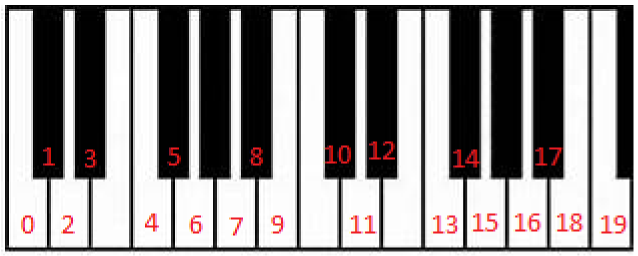


Figure 31 19-tone equal temperament layout

We plan to offer a different layout for people who want a model that makes sense musically. One idea for this model is to remove all format entirely, but this model does not reveal any clues to how the temperament functions. Instead, we could create an algorithm to generate a custom layout that is like a piano layout for every system but has a structure that fits a scale within that temperament. For an 8-tone equal temperament we could use a layout like the following figure.

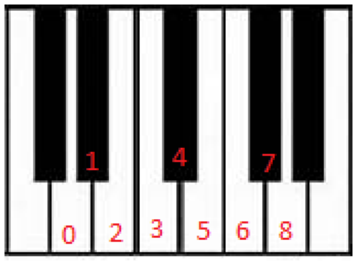


Figure 32 8-tone equal temperament custom layout

The equal temperament builder will allow users to build an equal temperament with formats other than 12-tones per octave. The builder allows the user to control the number of tones per octave, octave size, tuning center, tuning start, and end. The octave size changes how wide octaves are. The tuning center is the center frequency, the start is the first system integer, and the end is the last system integer. These parameters help resize the ATS so it fits into the MIDI format, depending on the piano layout, the tones will be trimmed out and keep the tuning center the center of the temperament.

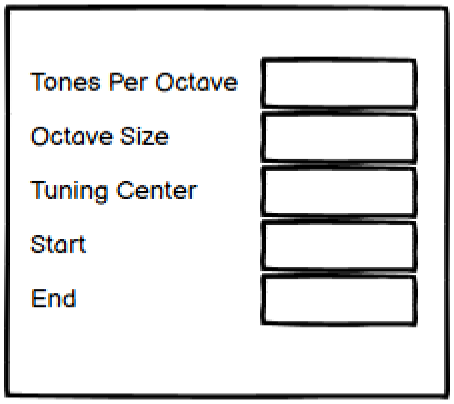


Figure 33 Equal Temperament Builder Wireframe

The builder will plug the values into the following formula. C is the tuning center, O is the octave size, t is the tones per octave, and s is the index or step that increments from start to end.

$$E_s = C * O^{\frac{s}{t}}, \qquad (s \mid start \leq s \leq end)$$

The builder will restrict the user from having an overlapping start and end, the start or end overlapping with the center, or entering a non-integer value. We may restrict the octave size and tones per octave to whole numbers or rational numbers above zero. The bounds of the ATS should fall within human hearing range, that is approximately 20 Hz to 20 kHz [42]. We can warn the user that the ATS is beyond human hearing range.

6.4 Class Diagrams

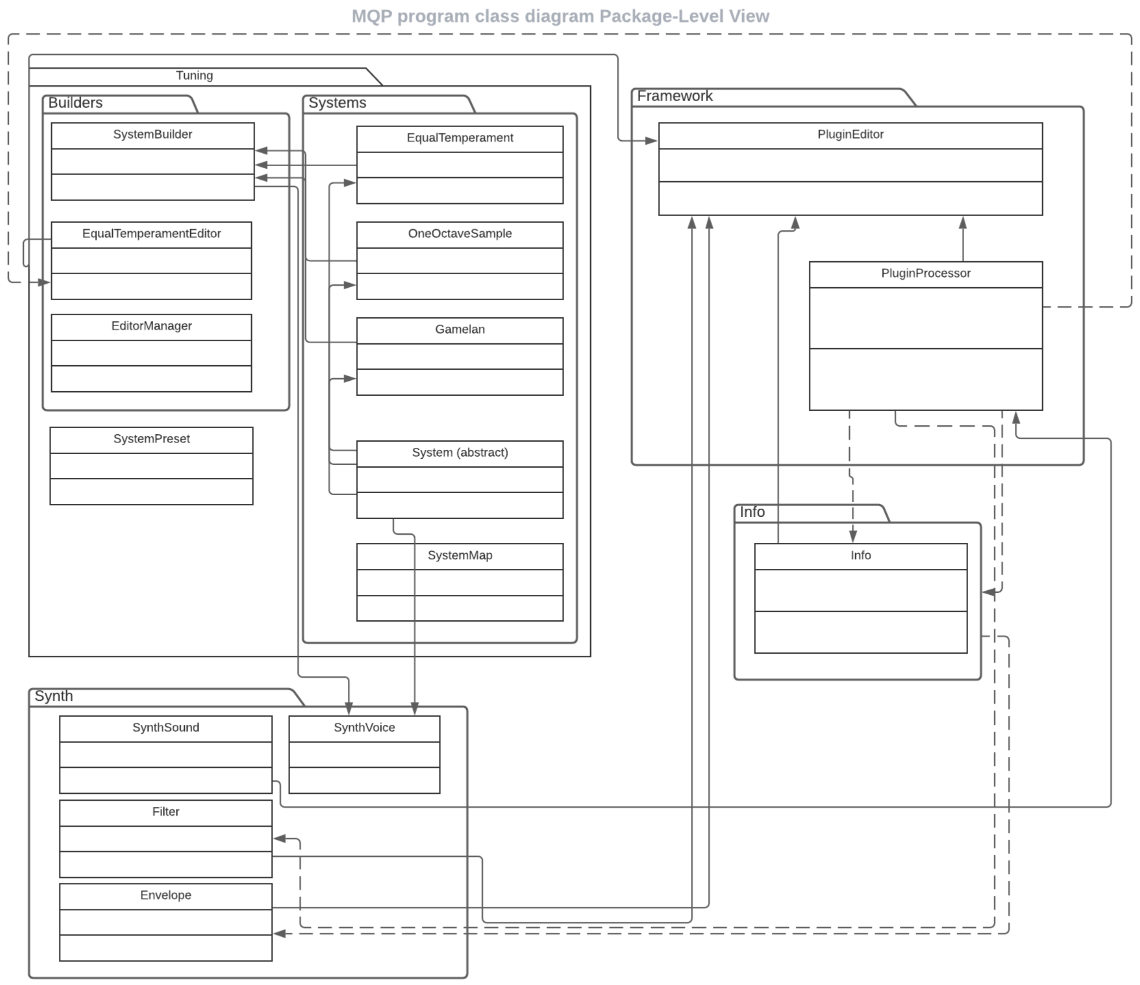


Figure 34 Package-Level Class Diagram

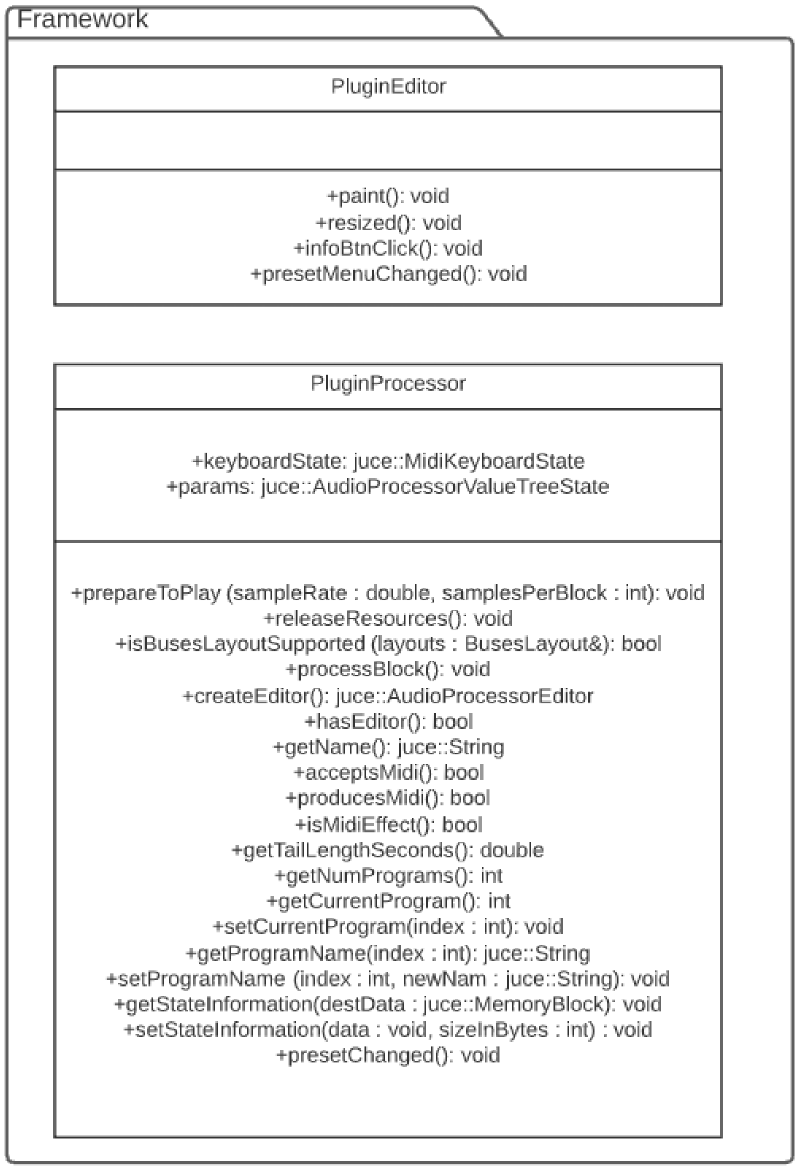

Framework
PluginEditor
+paint(): void
+resized(): void
+infoBtnClick(): void
+presetMenuChanged(): void
PluginProcessor
+keyboardState: juce::MidiKeyboardState
+params: juce::AudioProcessorValueTreeState
+prepareToPlay (sampleRate : double, samplesPerBlock : int): void
+releaseResources(): void
+isBusesLayoutSupported (layouts : BusesLayout&): bool
+processBlock(): void
+createEditor(): juce::AudioProcessorEditor
+hasEditor(): bool
+getName(): juce::String
+acceptsMidi(): bool
+producesMidi(): bool
+isMidiEffect(): bool
+getTailLengthSeconds(): double
+getNumPrograms(): int
+getCurrentProgram(): int
+setCurrentProgram(index : int): void
+getProgramName(index : int): juce::String
+setProgramName (index : int, newNam : juce::String): void
+getStateInformation(destData : juce::MemoryBlock): void
+setStateInformation(data : void, sizeInBytes : int) : void
+presetChanged(): void

Figure 35 Class Diagram for the framework package

The JUCE framework provides us with the “PluginEditor” and “PluginProcessor” classes. These are the two main “brain” classes that control everything from actual audio processing to the user interface. We had to build all our other classes to accommodate these two framework classes.

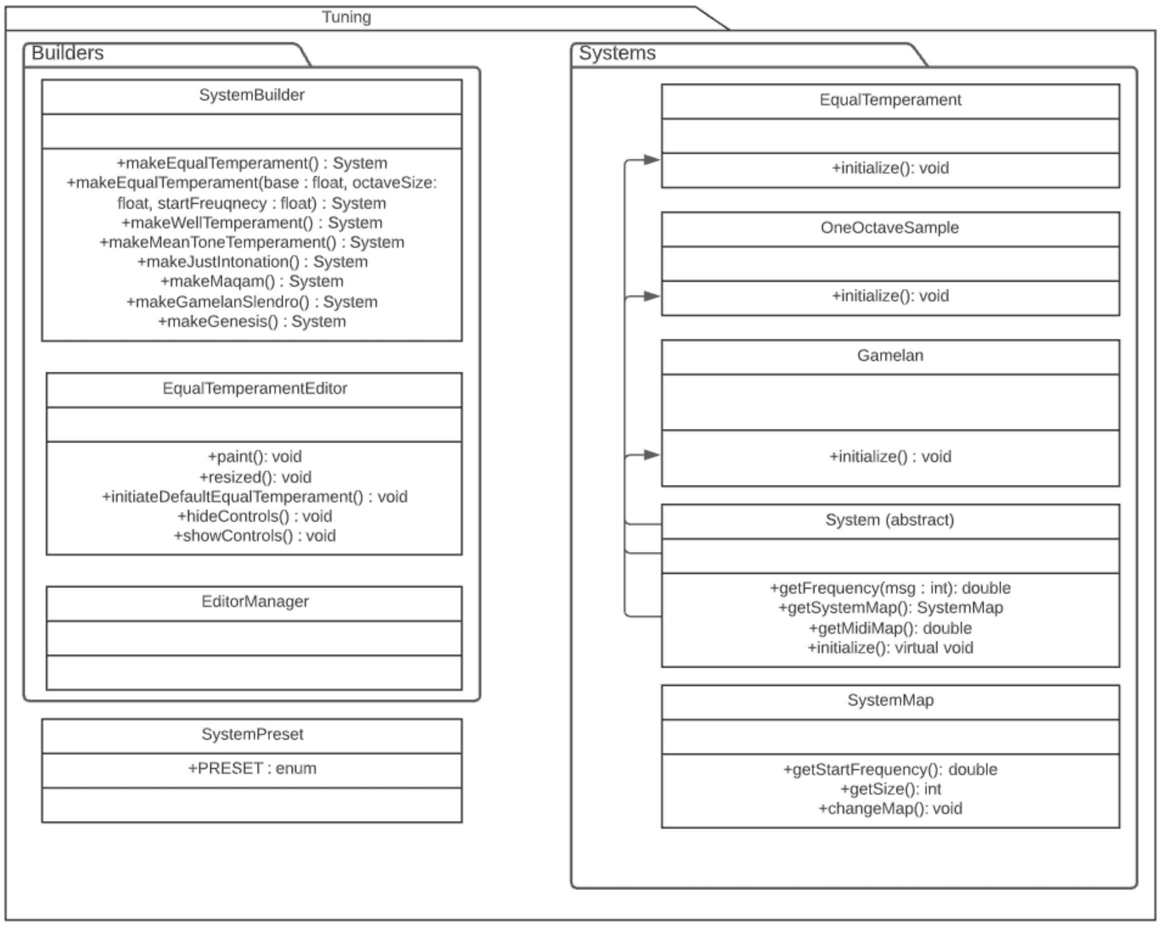

Tuning
Builders
SystemBuilder
+makeEqualTemperament() : System
+makeEqualTemperament(base : float, octaveSize: float, startFreuqnecy : float) : System
+makeWellTemperament() : System
+makeMeanToneTemperament() : System
+makeJustIntonation() : System
+makeMaqam() : System
+makeGamelanSlendro() : System
+makeGenesis() : System
EqualTemperamentEditor
+paint(): void
+resized(): void
+initiateDefaultEqualTemperament() : void
+hideControls() : void
+showControls() : void
EditorManager
SystemPreset
+PRESET : enum
Systems
EqualTemperament
+initialize(): void
OneOctaveSample
+initialize(): void
Gamelan
+initialize() : void
System (abstract)
+getFrequency(msg : int): double
+getSystemMap(): SystemMap
+getMidiMap(): double
+initialize(): virtual void
SystemMap
+getStartFrequency(): double
+getSize(): int
+changeMap(): void

Figure 36 Class Diagram for the tuning package

The System class is an abstract class that can be turned into all sorts of different tuning System subclasses. The classes “EqualTemperament” and “OneOctaveSample” are builder classes that can create almost any type of ATS, whereas Gamelan is its own class because that ATS has special attributes that cannot be given to it inside one of the builder classes.

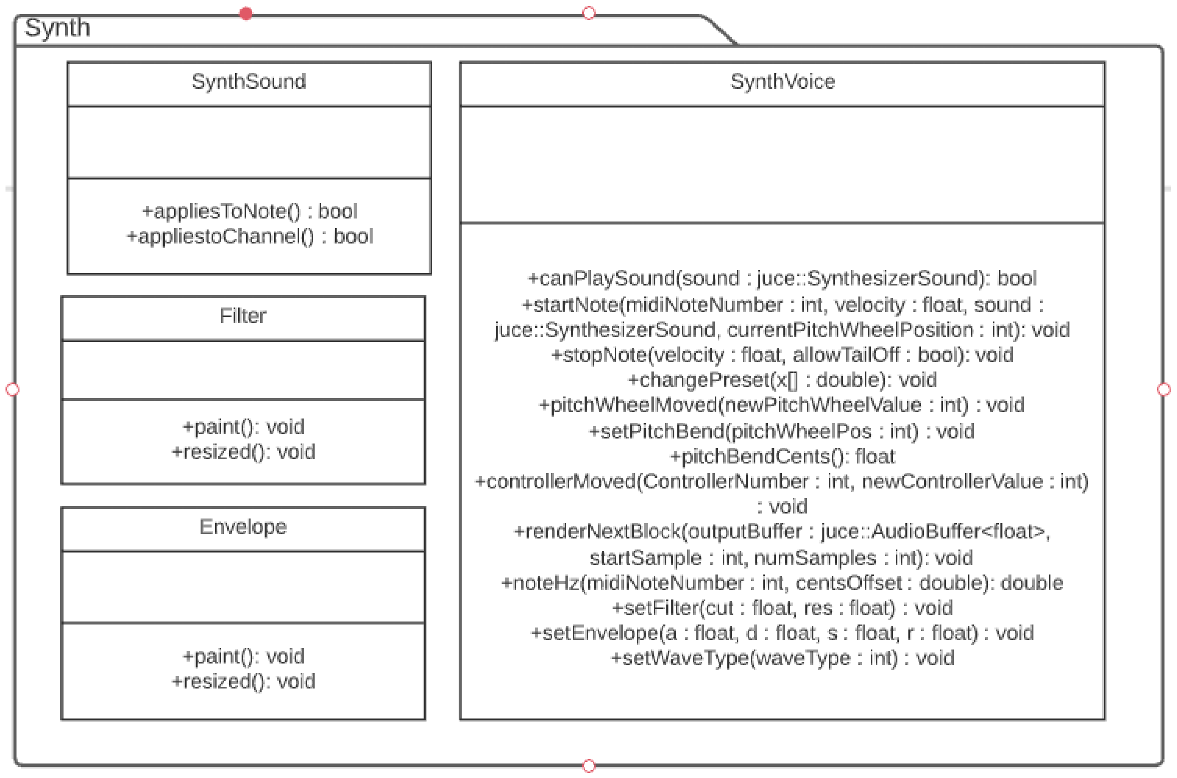

Synth
SynthSound
+appliesToNote() : bool
+appliestoChannel() : bool
Filter
+paint(): void
+resized(): void
Envelope
+paint(): void
+resized(): void
SynthVoice
+canPlaySound(sound : juce::SynthesizerSound): bool
+startNote(midiNoteNumber : int, velocity : float, sound : juce::SynthesizerSound, currentPitchWheelPosition : int): void
+stopNote(velocity : float, allowTailOff : bool): void
+changePreset(x[] : double): void
+pitchWheelMoved(newPitchWheelValue : int) : void
+setPitchBend(pitchWheelPos : int) : void
+pitchBendCents(): float
+controllerMoved(ControllerNumber : int, newControllerValue : int) : void
+renderNextBlock(outputBuffer : juce::AudioBuffer<float>, startSample : int, numSamples : int): void
+noteHz(midiNoteNumber : int, centsOffset : double): double
+setFilter(cut : float, res : float) : void
+setEnvelope(a : float, d : float, s : float, r : float) : void
+setWaveType(waveType : int) : void

Figure 37 Class Diagram for the synth package

The Synth classes handles all the sound design aspects such as which note to play and what the synth sounds like. The “SynthVoice” class specifically holds all the ATS we provide to the user so it can easily look up note information.

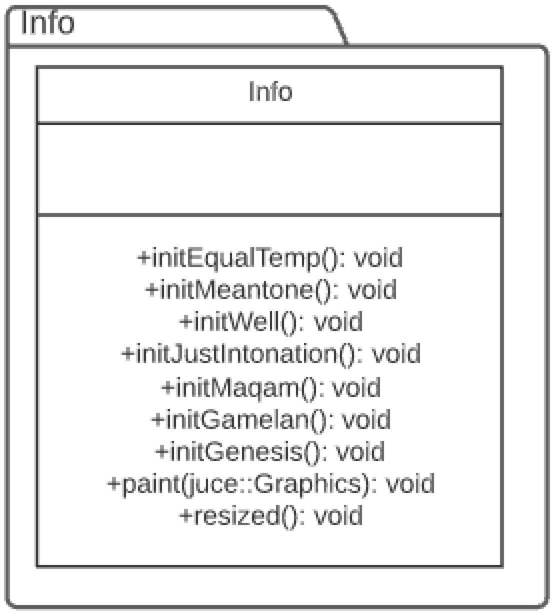


Figure 38 Class Diagram for the info package

The “Info”, “Filter”, and “Envelope” classes are all UI modules which get sent to the “PluginEditor” which handles creating the UI.

6.5 DAW Interaction

The JUCE framework allows us to build our plugin as a vst3, a format that DAWs can read. The only DAW interaction we needed to worry about was user-saved patches and how users can interact with the attributes in our plugin. Some DAWs allow users to “deconstruct” the plugin stripping away the UI and revealing sliders to control each attribute in our XML tree state so we needed to make sure the user cannot break the plugin through this method.

7 Software Development

7.1 Iterations

We had 6 sprints and iterations during the development process for our plugin. Each sprint lasted a week. We followed the Agile Scrum methodology and had daily Scrums and sprint retrospectives and reviews.

7.1.1 Iteration 1

Completed user stories:

- As a musician, I want to use equal temperament to create music
- As a musician, I want to be able to record my music in any ATS I select to integrate the new ATS into my music
- As a musician, I want to use my MIDI controller/keyboard to play tones from the current ATS to easily try out new ATS
- As a musician, I want the keys on the virtual keyboard to light up when I play them to give me clarity as to what note I am playing

Completed user story hours: 37

Our first sprint started off relatively slow. Before the sprint began, we had begun testing out JUCE and the other software we are using. Since neither of us had used JUCE before, it was very annoying to get everything set up the way it should be. It eventually worked out and we started our plugin with a very simple app that just says, “hello world”. We figured out that JUCE can generate both a .vst3 file and a .exe file when you build the code so we can test the app in a DAW and as a standalone app.

JUCE audio plugins start off with two distinct classes. The Editor and the Processor. The editor handles the entire frontend while the processor handles all the actual audio processing.

For our first sprint planning we decided that the main goal would be to create a simple plugin that allows us to play in equal temperament with a midi keyboard. Along with this, we wanted to get the on-screen keyboard to work.

Much of this sprint was simply getting used to JUCE and what it has to offer. We worked on creating a basic equal temperament synthesizer. We decided to use the “Maximilian” library for all the sound synthesis. The Maximilian library was hard to integrate into the code. However, we quickly figured out what specific files we need to import into our project.

Once Maximilian was integrated making the synth itself was easy. In the synth class there is also a section where we can easily change the frequency value of the note being played so integrating other ATS that are not equal temperament would not be too difficult.

We used the JUCE MIDI keyboard to add a simple keyboard to the screen. The keyboard works with clicking on the keys, pressing keys on a computer keyboard, or using a MIDI controller. Adding a keyboard was trivial, but we wanted to disable keys and control the layout of the keyboard. There was no interface to change the keyboard inside the class itself, so we created an external Keyboard Layout class to control specific aspects of the keyboard. Then we extracted some of the hard-coded aspects of the keyboard drawing process to disable keys. First, we tried just disabling the key drawing, but since the black keys are drawn on top of the white keys, there is no key where the black key is. Then we added a custom color for disabled keys, and an attribute to the key drawing methods, to tell the keyboard to draw the key grey if the key was disabled. This method made the keys act disabled and not respond to anything. Then we tried to add a custom layout to the keyboard, first we extracted out the arrays that ordered the black and white keys. These arrays seemed promising, but we found there were many other hard-coded drawing dependencies. Controlling two different arrays for one layout also seems impractical. To make the keyboard change layout easily, we would need to create our own keyboard class. One problem facing us was that we had no idea how to get the on-screen keyboard to play sounds when you click on the keys with a mouse.

Sprint Retrospective:

For our sprint retrospective we concluded that this sprint went well, we got more comfortable with JUCE and what it has to offer. We encountered some annoying aspects of JUCE but we knew we would get used to JUCE eventually. We also had some merging problems, so we are going to try to merge our branches together more often. We also needed to make sure we did scrum at least 5 nights in a week so we are aware of what each other are working on.

7.1.2 Iteration 2

Completed user stories:

- As a musician, I want to select a preset system so that I can use that ATS in my music in a DAW.
- As a musician, I want to use Maqam to create music
- As a musician, I want to use Gamelan to create music

Completed user story hours: 10

For our second sprint we wanted to integrate the preset ATS into the plugin. Since this is still week 2, we are still easing into JUCE and getting used to the framework. We ran into one of our first major problems this sprint. For some reason certain classes (that inherit JUCE classes) cannot add extra added methods such as getter and setters, especially the synth class. These classes did not operate in the standard way you would expect and therefore there are sometimes special ways to access them.

We worked on implementing the preset ATS this sprint. We initially started making the framework of how we are going to organize our class structure. In the design part of our project, we decided to make “system” a super class and have all our presets inherit “system’s” attributes. The most important aspect of these classes is their midi mapping array. This is an array of doubles, with 128 slots. There are 128 different MIDI note numbers so each slot in this array correlates to the frequency that should be played when that midi note number is called.

Some of the ATS we created were hard coded such as Maqam and Slendro with specific loops and starting frequencies. We decided to start adding flexibility to ATS, rather than have ATS be hard-coded. First, we created a system layout class to control system starting frequency, the start, center, and end of a system. We created an interface so systems can loop and fetch values easily. Later, we realized this interface was too complex and reduced it next iteration.

Going back to the extra method problem, we ran into the problem that we did not know how to give the synth class the current ATS we were using. After a bit of research, we discovered that the only way to access the synth class was by accessing the class through the process function in the processor class. This process function is the most important function in our code for audio synthesis, so it makes sense that we access the synth from that function.

After we figured out how to give the synth class information implementing the preset ATS was easy with a combo box (drop down menu) in the editor class.

We also decided that we wanted the user to be able to control certain sound aspects of the synth such as a filter. We are expanding our UI design to fit control knobs for certain parameters. We are not certain how many parameters to include at this time, but we added additional questions to our survey to see what people would be interested in.

We also worked on saving states of the plugin for DAW saved states. This is an important aspect for user to be able to save states of the plugin to access later. We did not get that far with this aspect, but we plan to give it a lot of attention in the future.

Sprint Retrospective:

For our sprint retrospective, we decided that this sprint also went well, although we did not get as much done as we would have liked. We wanted to have all the ATS implemented but that did not pan out. This was mainly due to not understanding how to feed the synth ATS values so we could not tell the synth what ATS to play. It took us a few days to fix this problem, so we did not have as much time to create the class architecture. Other than that barrier everything went ok, we were able to accomplish some of our sprint goals and we became a lot more confident with using JUCE this week.

7.1.3 Iteration 3

Completed user stories:

- As a musician, I want to control a filter for the synthesizer so that I have more sound design options available for creation.

- As a musician, I want to control the ADSR envelope for the synthesizer so that I have more sound design options available for creation.
- As a musician, I want to use Just Intonation to create music
- As a musician, I want to use Mean Tone Temperament to create music
- As a musician, I want to use Well Temperament to create music

Completed user story hours: 65

Last sprint we decided to implement synth controls and the rest of the preset ATS. When creating the synth controls, we found out an easy way of organizing large sections of the UI as their own class. This was done by making UI classes that inherit the component JUCE class. For now, we implemented a filter, with controls for cutoff and resonance, and an envelope, with controls for attack, decay, sustain, and release. These were simple to implement with only a few small bugs along the way. The ADSR/Envelope was not working the way we intended, we needed to do more testing and try to fine tune these controls. We also want to investigate implementing other effects such as modulation or reverb, but we are waiting for survey results to fully implement these aspects.

We tried to implement the info page, but we quickly learned that pop-up pages did not exist in the JUCE framework, so for next sprint we are going to find a solution for that. We were thinking of having the info button switch views between the main controls of the plugin, and an info section.

We also implemented all the preset ATS. We did this by creating a class framework that will also allow us to implement the builders in the future. Currently the only ATS that was not working was Pelog because there was not any concrete evidence to help us re-create the ATS.

Sprint Retrospective:

For our sprint retrospective, we thought that we made good progress on the UI, it was finally starting to look like a real plugin. Furthermore, a lot of the class structure were way more fleshed out now which is great. This aspect made creating the system builder easy. All the presets worked which is the most important part of our plugin. We implemented all the ATS we initially agreed on besides Pelog. We could not find any trustworthy evidence on frequency information on Pelog; therefore, we decided to remove Pelog from our plugin.

7.1.4 Iteration 4

Completed user stories:

- As a musician, I do not want any extra information to be obtrusive to the main UI so that I can ignore the extra info if I want
- As a musician, I want to control the waveform for the synthesizer so that I have more sound design options available for creation.
- As a musician, I want to be able to save my parameters as patches in DAWs, so I do not have to recreate patches every time I want to use them.

Completed user story hours: 30

For Iteration 4, we wanted to implement the information feature which will allow the user to understand a preset ATS before using it. We also wanted to implement the builder feature.

First and foremost, our survey has not been approved yet, this survey now has questions that will help guide the rest of our iterations. We have decided to go ahead and implement all the features discussed in the survey. When we get the survey results back, we will then decide if we need to remove or hide any features for the more advanced users to access.

We finished the framework for the Information screen this week. We originally planned to have this feature be in a pop-up window. However, we learned that JUCE is not the most conducive to pop-up windows. So, we settled on a ATS that will make the information screen visible by the click of a button. This button will also make some controls invisible to make room for the information screen. The button simply can swap between these two states. While we set up the framework, we did not fully fill out all the information in the section. Furthermore, we had troubles fitting the text. Sometimes, in JUCE, if you do not get give a text label enough space, it will start to squish. However, both problems should be easy fixes with time.

We also added waveform selection to the synthesizer. This is another sound design aspect that will give the user more options. The user can now select between a saw, sine, square, and triangle wave.

We were planning on implementing the builder this week, but we ran into problems. We originally wanted text entry boxes where users can enter parameter values for a builder, however we could not find a clean way to prevent the user from entering anything other than numbers. We started brainstorming and we thought up some new ideas for the builder, such as using sliders that correspond to values. So, all the user must do is adjust the sliders to create a system.

Sprint Retrospective:

For our sprint retrospective, we agreed that we did not hit our goals this week. This was due to many factors such as other class commitments and the break coming up.

However, on the bright side, we did not feel “stuck” on anything and knew we were able finish all our features, even if that meant working a bit during the winter break.

7.1.5 Iteration 5

Completed User Stories:

- None

Completed user story hours: 0

Iteration 5 occurred during spring break. Therefore, we were not able to make any big changes to the plugin. However, we did some planning for the next few months. We learned that C Term will not start until the end of January, so we plan to use the extra time to work and flesh out our plugin. Furthermore, since we could never find concrete interval values for the Pelog system, we decided to implement a new ATS in its place. We decided to try and implement Harry Partch’s Genesis Scale, a 43 note-per-octave scale. Historically multiple instruments were required to play a full octave of the scale, so we decided to implement it into our plugin so that a single instrument can play every note in the scale.

Sprint Retrospective:

For sprint retrospective there was not much to discuss since no programming occurred this week.

7.1.6 Iteration 6

Completed user stories:

- As a musician I want to use the Genesis Scale to create music.
- As a musician, I want to know the origin and brief history of the ATS I am using so I have some background knowledge before using a ATS.

- As a musician, I want to know how different aspects of the ATS function, and some basic music theory (if possible), so I can understand my musical options when using a ATS.
- As a musician, I want to create my own musical ATS from scratch to experiment with new ATS.
- As a musician, I want to create an equal temperament system that is not 12 notes to experiment with new ATS.
- As a user of the plugin, I want to be able to save my ATS to “patches” so I can use them as many times as I need without recreating them.

Completed user story hours: 123

For this sprint we added the Genesis scale and completed the information page functionality. The Genesis scale took quite a bit of research to fully understand how it works. Furthermore, implementing it was also difficult. We ran into many bugs along the way where the program would crash if you played a wrong note. In addition, sometimes when we made a change to the program and built it again, the change would not have been applied. We suspected this has something to do with memory leaks and read access violations, however, we eventually got the Genesis scale working.

On the other hand, we finished the functionality of the information aspect of our plugin. There still needs to be some micro-adjustments and making sure the information is presented in a concise and easy-to-understand way.

We also got the survey results back, which can be seen in the MQP Survey Results.csv file. The biggest takeaway from the survey was that musicians are split on the idea alternative keyboard layouts will help with understanding new systems. Since the basic features are still not fully working, we will ignore the alternative keyboard layout for not.

We investigated adding new sound design aspects to the plugin such as reverb or modulation. After looking at the survey results and doing some initial testing, we decided that these features are unnecessary, many musicians already have external effects plugins for effects like reverb or chorus.

We were behind on the ATS builder. We wanted to have it completed by this iteration, but we did not end up finishing it this sprint. We had all the background framework for the builder, so it was just a matter of connecting that framework with the UI and the xml tree state so users can save patches of their unique systems.

Sprint Retrospective:

For our sprint retrospective we were happy we implemented the genesis scale as we think it is an amazing ATS that many people will be interested in playing. We also thought that we made really good progress on the information feature and it was basically done at this point. However, we were disappointed that we were not able to finish the builder and alternative keyboard layouts by the end of the term. We decided to implement these two features before C Term starts so we did not have to worry about new features during testing.

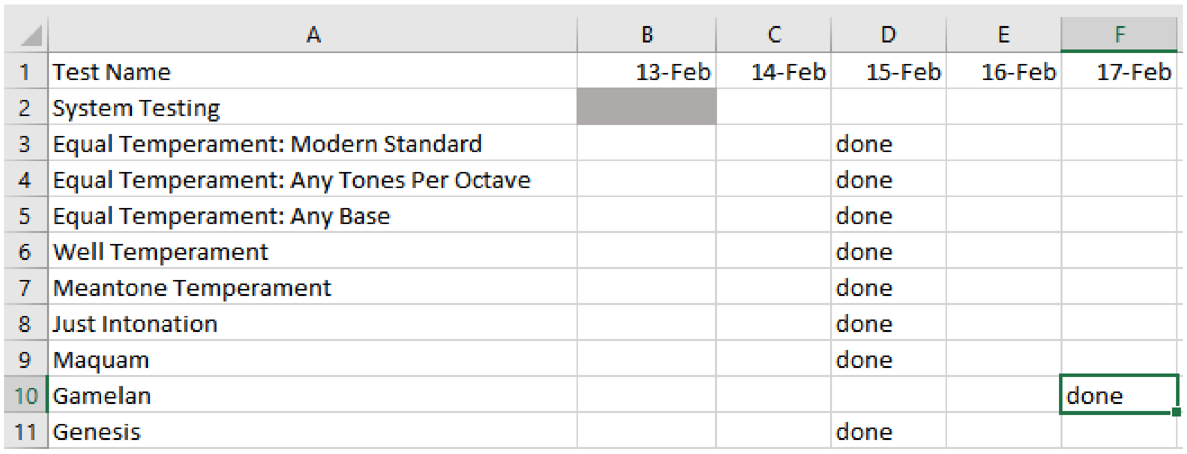
7.2 Testing Strategy

7.2.1 Unit Testing

Since our code relies on many UI components and JUCE framework there were not that many unit tests we could realistically make. Regardless, we wanted to heavily test our "systems" classes to make sure the ATS we are building are exactly how we want them to be. Furthermore, we added unit tests that are intended to try and break the ATS build so we can then add protections against breaking in the future.

We ran into multiple issues setting up the unit tests in the first place. We were new to unit testing in C++, so we were not sure how to create tests in our project or what framework to use. We eventually made a new Visual Studio solution purely for unit testing. We used the Microsoft Test framework to build and run the unit tests. We tested ATS classes by making sure each one produced the frequencies we expect. The ATS should function the same independent of the starting tone, and where we test the system. We tested the generation algorithm, rather than the ATS themselves, but we added some testing redundancy to test every system. Equal temperament systems have a consistent half step ratio between consecutive tones. We tested multiple classes of equal temperament to show that the ATS can build modern 12 Tone Equal Temperament, but it can also build the ATS with any number of tones per octave and any base. Meantone temperament, Well temperament, Harry Partch's 43 tone scale, and Just intonation were all created using a class that generates entire ATS based on a one octave sample. We tested each ATS based on the sample we gave it. Gamelan is 5-tone equal temperament, but it has a custom layout, so it had to be tested based on the layout, and that the ATS fit in the layout.

Our regression tests for unit testing are displayed in figure 39. All the unit tests passed by February 17th, 2021.



	A	B	C	D	E	F
1	Test Name	13-Feb	14-Feb	15-Feb	16-Feb	17-Feb
2	System Testing					
3	Equal Temperament: Modern Standard			done		
4	Equal Temperament: Any Tones Per Octave			done		
5	Equal Temperament: Any Base			done		
6	Well Temperament			done		
7	Meantone Temperament			done		
8	Just Intonation			done		
9	Maquam			done		
10	Gamelan					done
11	Genesis			done		

Figure 39 Unit Testing Records

7.2.2 Integration Testing

There was not a lot we could test through integration testing due to how the JUCE framework and various UI elements work. All the interactions between classes happens in the framework classes. The framework requires the application to be running. Therefore integration testing would have been incredibly difficult if not impossible to do so we decided to just stick with user-level and unit tests.

7.2.3 User-level Testing

For user level testing we decided to test all our use cases in and outside the DAW to make sure each of them worked as intended, with no bugs. We got off to a rough start with user testing because we ran into many problems when using the plugin inside a DAW, event though it was working perfectly as a standalone program. Debugging was incredibly hard when we used the plugin inside a DAW, because we could not set breakpoints or look at individual pieces of data to verify that they contained the data we expect. So, there was a lot of trial and error when it came to user-levels testing in the DAW. We eventually figured out that the DAW was not reading our ATS maps (the arrays where we store the frequency values for each system) properly. Therefore, we had to make big changes to the infrastructure of our program especially how the main processor class sends information to and from the synth.

Once we fixed the first bug, we noticed that the program was infinitely consuming memory which would eventually lead to crashes. We found this out looking at the process memory usage in Visual Studio. Even when the program was idle, the memory in use increases linearly.

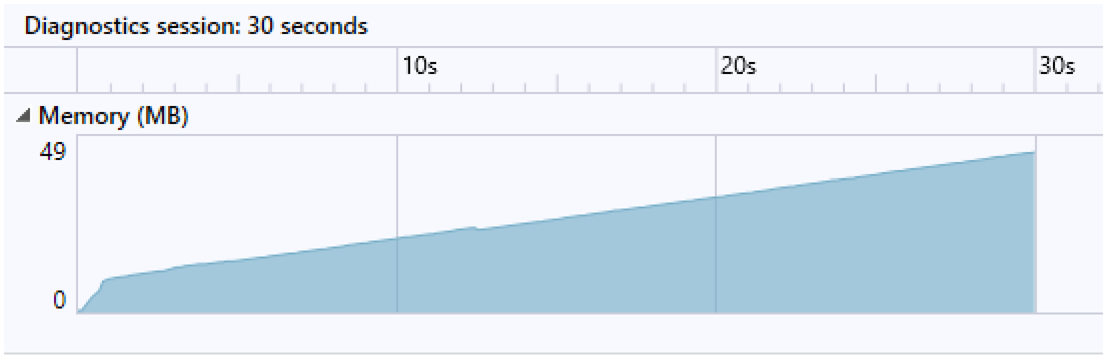


Figure 40 The program's idle memory consumption

We found that this problem was caused by a small part of changes we made to fix the first bug. Surprisingly, this bug was quite easy to squash out by not building a new ATS every time the user plays a note, but we also needed to accommodate for this change in multiple areas of our code. After the fix, the memory consumption does not increase over time, it fluctuates around 10 MB.

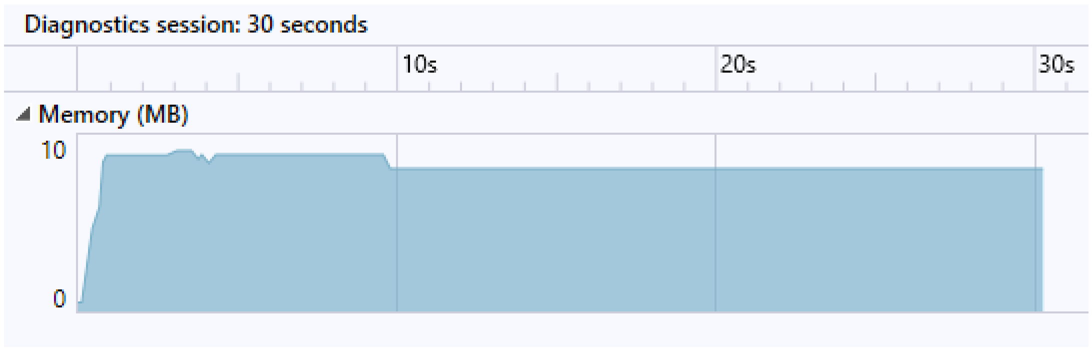
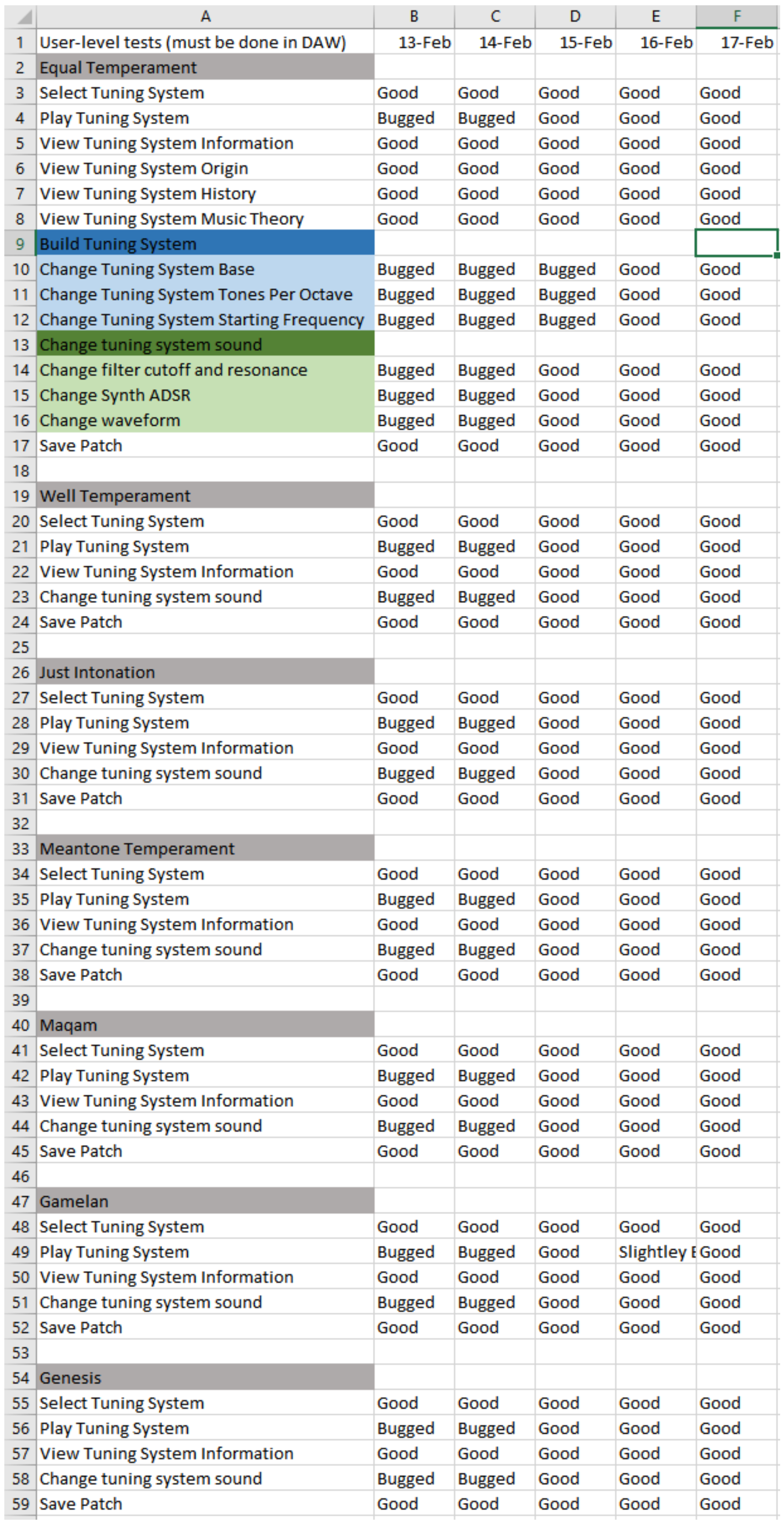


Figure 41 The program's idle memory consumption after the fix

At the end of user-level testing, we fixed all the bugs we found and the ones we knew about beforehand. We wanted to make sure that our plugin works seamlessly in and outside of a DAW so future users have no issues with crashes or ATS not sounding as they should.

Our user level regression tests are displayed in figure 42. The plugin was bug-free and complete on February 17th, 2021.



	A	B	C	D	E	F
1	User-level tests (must be done in DAW)	13-Feb	14-Feb	15-Feb	16-Feb	17-Feb
2	Equal Temperament					
3	Select Tuning System	Good	Good	Good	Good	Good
4	Play Tuning System	Bugged	Bugged	Good	Good	Good
5	View Tuning System Information	Good	Good	Good	Good	Good
6	View Tuning System Origin	Good	Good	Good	Good	Good
7	View Tuning System History	Good	Good	Good	Good	Good
8	View Tuning System Music Theory	Good	Good	Good	Good	Good
9	Build Tuning System					
10	Change Tuning System Base	Bugged	Bugged	Bugged	Good	Good
11	Change Tuning System Tones Per Octave	Bugged	Bugged	Bugged	Good	Good
12	Change Tuning System Starting Frequency	Bugged	Bugged	Bugged	Good	Good
13	Change tuning system sound					
14	Change filter cutoff and resonance	Bugged	Bugged	Good	Good	Good
15	Change Synth ADSR	Bugged	Bugged	Good	Good	Good
16	Change waveform	Bugged	Bugged	Good	Good	Good
17	Save Patch	Good	Good	Good	Good	Good
18						
19	Well Temperament					
20	Select Tuning System	Good	Good	Good	Good	Good
21	Play Tuning System	Bugged	Bugged	Good	Good	Good
22	View Tuning System Information	Good	Good	Good	Good	Good
23	Change tuning system sound	Bugged	Bugged	Good	Good	Good
24	Save Patch	Good	Good	Good	Good	Good
25						
26	Just Intonation					
27	Select Tuning System	Good	Good	Good	Good	Good
28	Play Tuning System	Bugged	Bugged	Good	Good	Good
29	View Tuning System Information	Good	Good	Good	Good	Good
30	Change tuning system sound	Bugged	Bugged	Good	Good	Good
31	Save Patch	Good	Good	Good	Good	Good
32						
33	Meantone Temperament					
34	Select Tuning System	Good	Good	Good	Good	Good
35	Play Tuning System	Bugged	Bugged	Good	Good	Good
36	View Tuning System Information	Good	Good	Good	Good	Good
37	Change tuning system sound	Bugged	Bugged	Good	Good	Good
38	Save Patch	Good	Good	Good	Good	Good
39						
40	Maqam					
41	Select Tuning System	Good	Good	Good	Good	Good
42	Play Tuning System	Bugged	Bugged	Good	Good	Good
43	View Tuning System Information	Good	Good	Good	Good	Good
44	Change tuning system sound	Bugged	Bugged	Good	Good	Good
45	Save Patch	Good	Good	Good	Good	Good
46						
47	Gamelan					
48	Select Tuning System	Good	Good	Good	Good	Good
49	Play Tuning System	Bugged	Bugged	Good	Slightley E	Good
50	View Tuning System Information	Good	Good	Good	Good	Good
51	Change tuning system sound	Bugged	Bugged	Good	Good	Good
52	Save Patch	Good	Good	Good	Good	Good
53						
54	Genesis					
55	Select Tuning System	Good	Good	Good	Good	Good
56	Play Tuning System	Bugged	Bugged	Good	Good	Good
57	View Tuning System Information	Good	Good	Good	Good	Good
58	Change tuning system sound	Bugged	Bugged	Good	Good	Good
59	Save Patch	Good	Good	Good	Good	Good

Figure 42 Regression Testing for User-Level Tests

8 Assessment

8.1 Discussion

Overall, we are happy with how our plugin turned out. We accomplished our original goal of allowing musicians to play in ATS. This was by far the most important part of the program. It is apparent that there were some features we decided not to include, however, there were also features we added along the way. At first, we were dead set on meeting our exact goals but along the way we grew to be more flexible and open to change. Currently, we think our plugin is very clean, usable, and tight. Most users will find some use in each of the features we included, the user interface is simple and intuitive, and it performs well inside and outside of a DAW.

8.2 Learning Experience

Working with JUCE, DAWs, and audio processing in general was an intimidating task for both of us. At the start we were uncomfortable with all the unique frameworks and architecture JUCE provides. Along the way we ran into problem after problem with not a lot of resources to help us since JUCE is a relatively niche framework. However, we managed to overcome this struggle to create a plugin we can be proud of and learn a lot about audio processing, and music theory along the way. One of the biggest takeaways from this project has been learning how much music, math, and programming intersect with one another. We often do not realize it, but digital music and sounds are often created entirely through math and algorithms, and these concepts are very important to computer science.

Furthermore, we learned a lot about working in a team of two to create a professional application. There is no hiding when the group is so small, each of us had to take responsibility for big parts of the program and we had to make sure they were done well. There were a lot of responsibilities shared between the two of us, many times we did what we assigned to ourselves, but the times we did not were still useful for retrospection.

On the other hand, we further increased our skills with the Scrum methodology. At times it was hard to keep up with the daily meetings, and we always saw dips in productivity when we forgot about the meetings. We learned that the more communication we had between ourselves, the more our productivity and efficiency grew. We also learned to not just go through the motions of scrum, but take the meetings, reviews, and retrospectives seriously to get the most value out of them.

Overall, we are incredibly happy with our program and proud of the growth we have shown over these three terms. Furthermore, the whole process of creating an application out of an idea has been a great learning experience for the both of us and hopefully the lessons we learned will aid us in the future.

9 Future Work

We wanted to make a VST plugin that feels complete, that someone could download and start using, and I feel like we accomplished this goal. Despite this, there are many additions or changes one could do to this program to make it more useable or accessible.

There are some features that we discarded to make sure the core of our project was solid. These features, such as alternative keyboard layouts (section 6.3), freeform builders, one octave sample builders, and extra synth effects are all features that will improve the application (in our minds). Sadly, these features were not implemented due to time constraints.

Furthermore, we feel that the clarity of the plugin could be improved. There are instances where a user may be confused on the differences between some ATS, or which note is assigned to what key in others, or on the real difference between ATS. The information on ATS may be useful, but words alone cannot demonstrate system functionality or describe how one would create music with a ATS. Creating helpful interactive audio vis to demonstrate ATS layouts or theory may help make ATS more approachable. Keyboard layouts that properly reflect the structure of a ATS may help users experiment with unfamiliar ATS.

There is the option to include more sound-production features, or to create a more unique synthesizer from scratch. We used a synth to produce all our sounds, but we could have used a sample library, and altered the pitch of those samples, or allowed the user to upload a sample library to use as the sound production. We could have made the application an effects plugin, that alters the audio output of any VST to correct pitches based on input MIDI values. There are already excellent means to produce sounds, but not as many means to control the pitch of those sounds.

10 Conclusion

We started this project with a simple goal, we wanted to allow musicians to use and experiment with alternative musical temperaments and tuning systems. We decided to accomplish this goal by creating a VST plugin that will be compatible with any DAW. VSTs are very easy to use along with the fact that many musicians nowadays create music inside of a DAW. Our final plugin allows musicians to experiment with different ATS along with many other features dedicated to sound exploration and design.

Ultimately, we learned a lot about audio and sound programming as well as learning about the process of creating a professional application. There were struggles along the way as we had many issues with our framework and found bugs that seemed unfixable at the time. But in the end, we are both proud of our plugin and the work we put into it. Hopefully, the lessons we learned and skills we gained will benefit us in future as we continue our computer science careers.

11 References

[1] Britannica Equal temperament (2020).
[2] Reina, V. Music Theory Terms To Know – Basic & Advanced (May 31 2019 2019).
[3] Britannica Octave (2020).
[4] Record, T. J. A Reasoned Cacophony: Ornette Coleman - "Free Jazz" (2015).
[5] Kapoor, V. Jimi Hendrix Didn't Study Music Theory – why should I? (June 8 2018 2018).
[6] MTU Making Sense of Cents (2020).
[7] Britannica Timbre (2020).
[8] Sethi, R. Top 12 Most Popular DAWs (You Voted For) (2017).
[9] Jorgensen, O. H. *Tuning, The Perfection of Eighteenth Century Temperament, The Lost Art of Nineteenth Century Temperament, and The Science of Equal Temperament*. Michigan State University Press, 1991.
[10] Fonville, J. Ben Johnston's Extended Just Intonation: A Guide for Interpreters. *Perspectives of New Music*, 29, 2 (1991), 106-137.
[11] BabaYagaMusic Eastern Music Tone Production (2019).
[12] taq.sim Guide to Middle Eastern Modal Music and the World of Makams (2020).
[13] Britannica Gamelan Indonesian orchestra (2020).
[14] Braun, M. The gamelan pelog scale of Central Java as an example of a non-harmonic musical scale (2002).
[15] Duimelaar, P. Pitch & tuning (2003).
[16] Kendall, E. C. C. a. R. A. On the Tuning and Stretched Octave of Javanese Gamelans (1994).
[17] Plogue Chipsound (2020).
[18] Johnson, A. K. microsound. un twelve. http://untwelve.org/microcsound>
[19] Rationale. Source Forge. http://rationale.sourceforge.net/>
[20] Coul, M. O. d. Scala. Huygens Fokker. 3 July 2020.<http://www.huygens-fokker.org/scala/>
[21] Mutabor. Technische Universitat Dresden. Tueday, October 7, 2014.<http://www.math.tu-dresden.de/~mutabor/index.html.en>
[22] Kritov, A. Sound Modeler. Sound Modeler. 2020.<https://www.soundmodeler.com/>
[23] CSound. CSound. 14 Aug 2020.<https://csound.com/>
[24] Braunstein, M. Offtonic Microtonal Synthesizer. Offtonic. https://offtonic.com/>
[25] Sevish. Scale Workshop. https://sevish.com/scaleworkshop/index.htm?name=Slendric%5B11%5D&data=202.12%0A233.69%0A435.81%0A467.37%0A669.50%0A701.06%0A903.19%0A934.75%0A1136.87%0A1168.44%0A2%2F1&freq=440&midi=69&vert=-1&horiz=2&colors=white%20black%20white%20white%20black%20white%20black%20white%20white%20black%20white%20black&waveform=triangle&ampenv=organ>
[26] Yi, S. blue. kunst musik. 2020.<https://blue.kunstmusik.com/>
[27] Hex. Dynamic Tonality. http://www.dynamictonality.com/hex.htm>
[28] John. Schismata. Schismata. http://schismata.net/>
[29] Walker, R. Tune Smithy. Robert Inventor. 2008.<http://robertinventor.com/software/tunesmithy/music.htm>
[30] Monzo, J. tonescape. tonalsoft. 2005.<http://tonalsoft.com/tonescape.aspx>
[31] Melodyne 5. celemony. 2020.<https://www.celemony.com/en/start>
[32] Tutorialspoint SDLC - Waterfall Model (2020).

[33] Powell-Morse, A. Waterfall Model: What Is It and When Should You Use It? (2016).
[34] Tutorialspoint SDLC - Agile Model (2020).
[35] L.inchpin A Beginner's Guide To The Agile Method & Scrums (2020).
[36] cprime WHAT IS AGILE? WHAT IS SCRUM? (2020).
[37] Scrum.org The Home Of Scrum (2020).
[38] Jira Software. 2020.<https://www.atlassian.com/software/jira>
[39] git. 2020-07-27.<https://git-scm.com/>
[40] Bit Bucket. 2020.<https://www.atlassian.com/software/bitbucket>
[41] Source Tree. 2020.<https://www.atlassian.com/software/sourcetree>
[42] McKechnie, J. Human Hearing Range. Hearing Direct.
9/7/2018.<https://www.hearingdirect.com/blog/human-hearing-range.html>

12 Appendices

A. Survey Questions

How Old Are you 18-30, 31-40, 41-50, etc.

- Below 18 (survey will end if they click this option)
- 18-30
- 31-40
- 41-50
- 50+

How would you describe yourself as a musician?

- Beginner (Just started learning within the past year)
- Intermediate (A couple years of playing experience)
- Experienced (Many years of playing experience)
- Expert (Multiple years of high-level training and experience)
- Professional (Music is your job)

On a scale of 1-5 (1 being no experience, 5 being an expert), how much do you know about music theory

- 1
- 2
- 3
- 4
- 5

In musical tuning, a **temperament** is a tuning system that decides the specific tone of each note. For example, 12 note equal temperament defines the A4 note at a frequency of 440hz.

Have you played in anything other than 12 note equal temperament?

- Yes
- No

On a scale of 1-5, how willing would you be to try out alternative temperament systems?

- 1 (Very Unwilling)
- 2 (Somewhat unwilling)
- 3 (Neutral)
- 4 (Somewhat willing)
- 5 (Very willing)
- I do not have enough information to answer the question

Are you interested in answering questions about a VST plugin that will help musicians play in alternative temperament systems?

- Yes
- No

Which of these settings would you be most interested in being able to access for sound synthesis?

- Synth waveform
- Attack/Decay/Sustain/Release
- LFO/Modulation
- Time effects (reverb, delay)
- I do not care about the sound synthesis

How interested would you be in a feature that allows users to create their own temperament systems?

- 1 (Very Uninterested)
- 2 (Somewhat uninterested)
- 3 (Neutral)
- 4 (Somewhat interested)
- 5 (Very Interested)

If you were to create an alternative music temperament what would you prefer?

- System Builder - generate a system based by entering system parameter values
- Freeform creation - enter individual frequencies for each tone

Which of the following do you think you would need to use another system in your music?

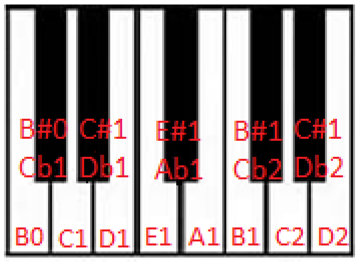
- ❏ History and background
- ❏ Musical theory
- ❏ The math used to create a system
- ❏ The system’s musical properties
- ❏ Sample pieces from the culture that created the system

❑ No information needed

Many alternative temperament systems do not conform to a standard keyboard layout. How comfortable would you be comfortable playing on a piano layout where some notes are disabled (due to the alternative system).

- 1 (Very uncomfortable)
- 2 (Somewhat uncomfortable)
- 3 (Neutral)
- 4 (Somewhat comfortable)
- 5 (Very comfortable)

How helpful would a custom keyboard layout be to help you understand and learn different systems? For example, if there were 8 notes per octave instead of 12, the layout may look like this.



- 1 (Very unhelpful)
- 2 (Somewhat unhelpful)
- 3 (Neutral)
- 4 (Somewhat helpful)
- 5 (Very helpful)

Please see the file, MQP Survey Results.csv for the survey results.