GENERATIVE MUSIC APPLICATION

PROJECT REPORT

Declan Kehoe

Supervisor: Professor Sean Bechhofer

MANCHESTER 1824
The University of Manchester

Abstract

Generative music describes an indirect form of music creation, wherein the composer sets parameters by which the music will generate itself.

The main goal of this project was to create an application that would allow a user to make generative music in real time, without overwhelming them with complex terminology, intimidating them with alienating interfaces, or confusing them with excessive detail.

The secondary goal was to fully leverage the language and IDE the application was built in, Processing (specifically Processing 3), by utilising as few external libraries as possible. This was decided so as to demonstrate the level of functionality that can be created 'from scratch' to new developers using Processing, and to reduce the complexity in sharing the project with others.

Overall, as this project is rooted in a human computer interaction student's perspective, the focus was on creating a set of user experience outcomes; for app users, and potential Processing developers, rather than a radical technological exercise.

This report will show how the project was carried out to achieve its stated goals using explanations of the research and development process, feedback from users, and self-reflection framed by the objectives.

Acknowledgments

I’m very grateful to my project supervisor, Sean Bechhofer, who was a patient and thoughtful voice throughout. Through discussions and questions he helped to define and establish what would be the criteria by which this initially abstract project could be corralled, and enabled a path for the development to take.

The various people who were sent multiple versions of the app, at many levels of unsightly and unfinished, but who still provided such insightful and useful feedback, were invaluable to the completion of this project.

Finally my partner, my parents, and my pets; without whom there’s no way I would have gotten through the experience of uni as a mature student. They are the best, and I’m lucky to know them, let alone be loved and supported by them.

The Impact of COVID-19

It is impossible to not also bear in mind the impact the COVID-19 pandemic has had on my ability to work on, and complete this project. While I was incredibly fortunate to have neither myself nor my close-family catch the disease, the reality of around 18 months of constant isolation for the sake of shielding my vulnerable family was inevitably a strain. I was lucky to be in an environment where I could continue to work, but the loss of interaction and academic support that the presence of other students brings was sorely felt.

Overall, the nature of my project, the privilege of my circumstances, and the supportiveness of my supervisor and family meant that the nightmarish whirlwind of the pandemic could blow through my life, without leaving too much damage.

Table of Contents

Abstract 1

Acknowledgments 2

The Impact of COVID-19 2

Table of Contents 3

List of Figures 5

1 - Introduction 6

- 1.1 Objectives and Assessment 6
- 1.2 Context and Project Differentiation 7
- 1.3 Report Structure 7

2 - Background and Processing 8

- 2.1 What is Generative Music? 8
- 2.2 State of the Art 9
- 2.3 Processing 3 - Quirks and Considerations 11

3 - Planning and Development 13

- 3.1 Splitting up the task 13
- 3.2 The Core Design 13
- 3.3 Project Timeline 14
- 3.4 Milestones 15

4 - Sound Subsystem 17

- 4.1 Sound, Ratios, and Scales 17
- 4.2 Final Note Implementation 18
- 4.3 Waves and Envelopes 18
- 4.4 The Oscillator Method 19

5 - Generative Engine 21

- 5.1 Early Architecture 21
- 5.2 Creating the Timeline 21
- 5.3 Finalising Functionality 23
- 5.4 Errors and Weaknesses 25

6 - GUI 27

- 6.1 Prototyping 27
- 6.2 Final Implementation 30

6.3 Limitations 31
7 - Evaluation 32
7.1 Semi-Structured Interview Questions 32
7.2 Key Themes from Interviews 32
7.3 Specific Improvements from Interviews 35
7.4 Self-Reflection 36
8 - Conclusion 39
Glossary 40
References 41
Appendix 42
Interview 1 Transcript - Participant P 43
Interview 2 Transcript - Participant S 45
Interview 3 Transcript - Participant K 47
Interview 4 Transcript - Participant M 49
Interview 5 Transcript - Participant E 50
Interview 6 Transcript - Participant N 53

List of Figures

1 - Visualisation of generative tape loops (Parviainen, 2017) 8
2 - Three screenshots from Trope (Eno, Chilvers, 2009) 9
3 - Screenshot of SuperCollider (Zhagun-Linnick, 2020) 10
4 - Screenshot of Nodal 2.0 (SensiLab, 2021) 11
5 - Screenshot of project code in Processing IDE 12
6 - Timeline of project development 14
7 - Visualisation of application architecture 22
8 - Screenshots of early prototype GUIs 27
9 - Screenshots of mid-point prototype GUIs 28
10 - Screenshots of the final GUI 29
11 - Screenshot of the interactive demo in Sketch 30
12 - The buttons created for the GUI 30

1 - Introduction

1.1 Objectives and Assessment

The primary objective of this project was to design, develop, and evaluate an application that allowed users to create generative music, in Processing's interactive development environment (IDE). Subgoals were also introduced to more broadly evaluate the success of the project. As such, the objectives for the project were defined as:

1. *A functional application must be built, that can be used to create generative music.*
2. *A minimal use of non-native libraries should be used for development, to better allow new Processing developers to understand and extend the code, and to personally develop a stronger grasp of Processing's capabilities.*
3. *The user experience should not be intimidating, instead welcoming users regardless of their previous experience with either generative music, music composition, or music creation applications generally. The app should also optionally provide information to more curious users as to how the generative aspects function more theoretically.*

Whether or not the project has achieved these goals will be ascertained differently for each objective. The primary assessment method for each goal is therefore:

1. The existence of the application itself is the most trivial to demonstrate, as it is a software artefact that either functions, or does not. The definition of creating generative music is somewhat more difficult to define, but this report will detail the definitions, and evaluate the app's success in this context.
2. The success of the second objective depends on the strictness of the definition of 'minimal use of non-native libraries'. In the case of this project, successfully meeting this goal would mean a developer can download the basic Processing IDE, open the source code file, and immediately run the app without downloading or configuring extra libraries. The code itself is designed to be readable, understandable, and extensible even to relative newcomers to either Processing specifically, of software development generally. As such, wherever possible, any specific software engineering solutions should forego complexity for the sake of expediency, in favour of clarity for the sake of legibility.
3. Semi-structured interviews with multiple participants, questioning their user experience with the app will be conducted after the development phase to ascertain the success of the third objective. The participants should have a range of previous exposure to music composition and music software to better judge the potential experience of multiple types of user.

As success, or, failure was binary only in the case of the first two objectives, a qualitative appraisal approach was chosen over a quantitative one for the third objective due to the subjective nature of defining its success.

I would also encourage the reader to try the application themselves, if possible, to better assess their agreement or disagreement with the conclusions found in this report.

1.2 Context and Project Differentiation

Generative music was a term coined initially to define a way of creating ambient, or background, music (see 2.1 for more on the history of generative music). Since the inception of this idea in the late 1980s, there have been multiple hardware and software solutions designed that enable a user to employ generative music techniques. While a hardware-based implementation was beyond the scope of this project, examples of this type served as a useful inspiration. Primarily however, the research phase of the project focused on generative music software applications.

Overall, the current generative music landscape ranges from the extremes of completely novice-friendly applications with high levels of abstraction, through to low level technical solutions aimed at generative music composers. For example, there are entirely touch-control and graphical user interface (GUI) driven applications which represent their sounds as lines and blobs of colour, and all musical choices were made by the application's creators during development. At the other end of the complexity spectrum are the software libraries which allow every parameter of the generative music to be set and modified by a composer, through their IDEs as part of software or music composition projects. The software was examined to gain a sense of the level of abstraction, interface complexity, and underlying compositional choices that had been previously attempted and accepted (see 2.2 for more on this research).

There is, however, little in the way of applications for generative music at a middling level of complexity; a generative music application that abstracts away intimidating details for the sake of newcomers, but can optionally allow a curious user to see some of the inner-workings. Allowing them to know what is driving the sounds they hear, and to help them understand how they are interacting with the process of generative music. This project aims to work in that middle ground.

1.3 Report Structure

This report is split into 8 sections, and their contents is as follows:

- The first section is this introduction.
- Section 2 discusses the background of generative music, the current state of the art in the field, and the way Processing's IDE design will be incorporated into the project.
- Section 3 presents the core project design, the development plan, and a timeline of the project's milestones.
- Sections 4 - 6 each describe one of the three subsystems that comprised the core design, and the specifics of their development.
- Section 7 evaluates the project's achievements compared to its three objectives (described in section 1.1), through excerpts of evaluative interviews, and self-reflection.
- Section 8 is a conclusion and summary of the project as a whole.
- There is also a glossary and index found at the end of the report. The glossary covers mostly the music theory and synthesis terminology used throughout this project. The index contains transcripts of all the evaluative semi-structured interviews conducted with participants, and the reader is encouraged to browse them for a more complete picture of the project's evaluation.

2 - Background and Processing

2.1 What is Generative Music?

Generative music is a term and technique for partially automating the process of music composition, popularised mostly by the composer Brian Eno. Although certainly not the first or only practitioner - Herremans et al., (2017) in fact trace the automation of composition all the way back to 1757, when a musical game based on dice rolls was released - Eno's work is certainly the most prominent influence on this project specifically, and on the development of generative music, generally. A useful way to understand what generative music is can be found through comparing it to traditional forms of music composition and in a talk designed to introduce and explain the history of generative music, Eno did so:

"Classical music, like classical architecture, like many other classical forms, specifies an entity in advance and then builds it. Generative music doesn't do that, it specifies a set of rules and then lets them make the thing."

— Brian Eno (1996)

This means that a 'composer' in a generative music context is not defining the progression of notes and chords directly. Rather, they are establishing a framework that the sounds will follow, and allowing the music to be whatever arises from adhering to those rules.

Generative music was utilised by Eno for his seminal 1978 album Ambient 1: Music for Airports. The specific technique began by recording a sound - a simple note, chord, or melody - on to magnetic tape as found in reel-to-reel tape machines. A section of tape including the recorded sounds would then be cut from the main tape and have its ends spliced together to form a loop. This loop of tape would now, when run through a player, continue infinitely, with the length of time to repeat the loop dictated by the physical length of the tape. Multiple loops of various lengths, each featuring recordings composed to be musically similar - so as not to be discordant when together - were set to play in different groupings. Once this system of loops has been set up, the different recording lengths, and different groupings of those recordings, lead to unpredictable and ever-changing interplays between the sounds (Eno, 1996). Due to the way the recordings were composed though, the chaotic outcome is still recognisably 'musical', and pleasing to the ear. The process behind the creation of this album is a clear example of generative music, and variations with different technologies are found in almost all generative music projects still.

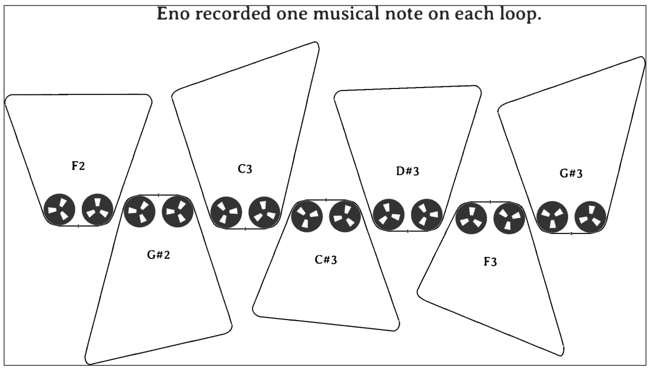
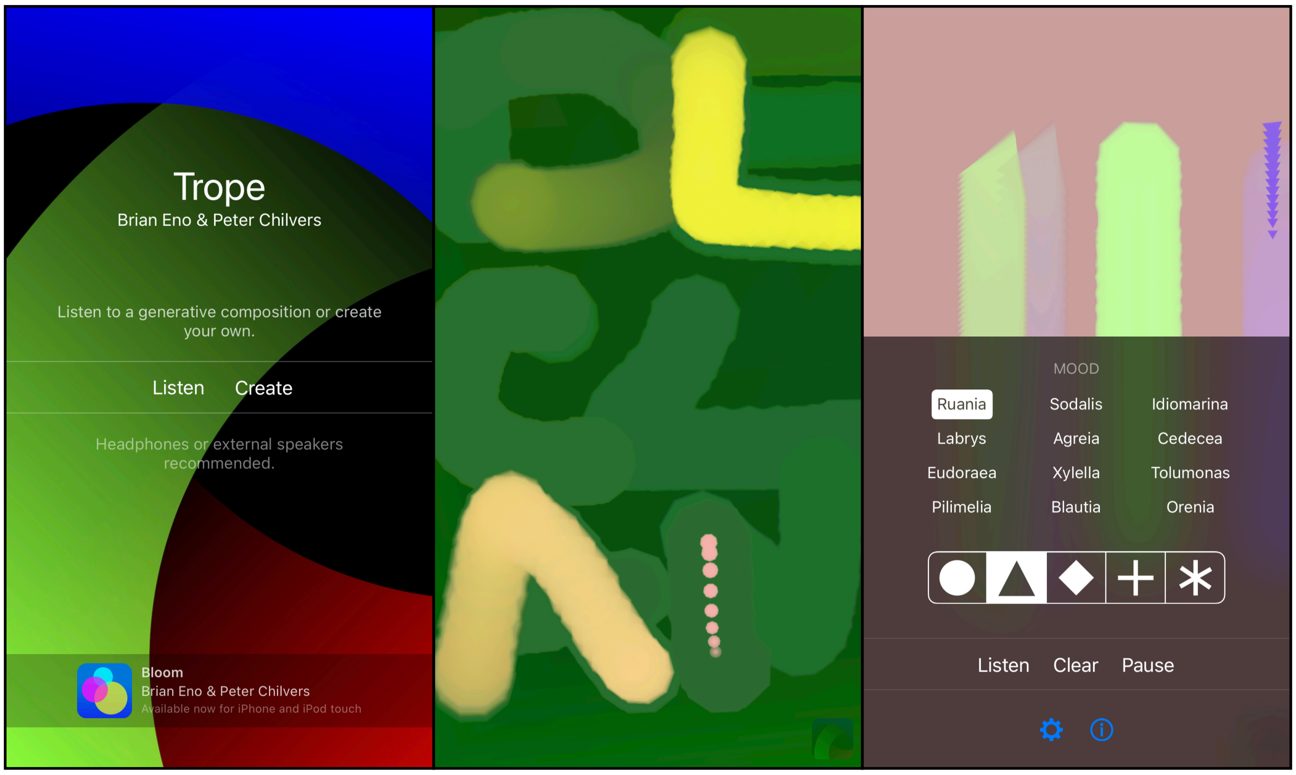


Figure 1 - A visualisation of the tape loops that were used on one track during the creation of Music For Airports (Parviainen, 2017)

The concept of changing and interacting loops of simple sounds is also the central idea behind the operation of the application developed for this project. The process of creating the rules that drove the sounds that would be looped, and how the user of the application could in turn interact with those rules, form the bulk of this report's contents.

2.2 State of the Art

A great many different ways to interact with and create generative music already exist. Again, Brian Eno's influence looms large, with iOS apps created by him in collaboration with musician and developer Peter Chilvers being some of the most widely accessible interactive forms of generative music. Early apps such as Bloom and Trope are highly interactive and influenced by the user, but those developed since (including Scape and Brian Eno: Reflection) take a more authored approach to presenting generative music. Specifically, the later apps instead of having user interaction most affecting the sounds, have the mix and melodies of composed sounds within them progress autonomously and endlessly.



Listen to a generative composition or create your own.

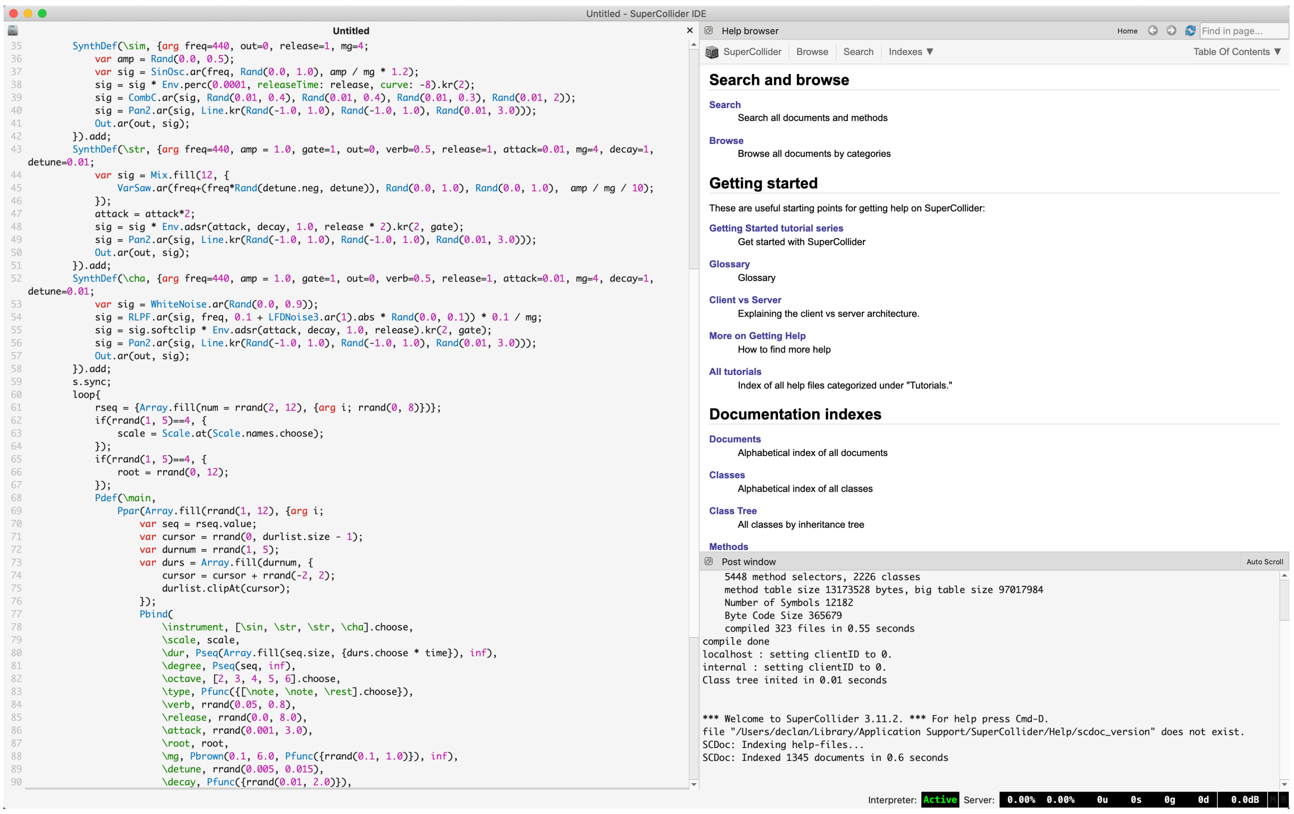
Headphones or external speakers recommended.

Figure 2 - Three screenshots from Trope, showing the screen upon opening the app (left), after drawing lines using a finger (centre), and the options shown when clicking the small logo at the bottom right of the screen during use (right) (Eno, Chilvers, 2009)

An application that was studied more deeply for this project was the early generative music iOS application; Trope, created by Eno and Chilvers in 2009 (see figure 2, above, for screenshots). Any instrumentation and musical choices were made entirely by the developers, and interaction with the app gives an abstracted version of control as to how those chosen sounds are played back. There is also an option for a user to 'listen', wherein a composition is generated autonomously from the predefined sounds. The relationship between the shapes drawn by the user on the screen and the sounds they trigger are unclear, except for longer lines seemingly equating to generally longer sounds. The delay between the loops of sounds heard, and the tone and timbre of those sounds, are not directly controllable. The shapes and 'moods' shown in the options screen are the closest the user gets to direct control, but are again deliberately abstract and not directly tied to recognisable musical concepts.

Trope's level of abstraction goes beyond the aims of this project, as one of the stated goals is to allow a curious user to understand some of the musical theory at play during music generation (1.1). However, the approachability of the interface, and the immediate positive feedback of hearing pleasant sounds resulting from your own interaction is gratifying, and something this project's application aims to emulate.

On the other end of the complexity spectrum to the apps of Eno and Chilvers is the use of music creation focused IDEs to develop generative music. One popular open source music-based IDE and language is SuperCollider (see figure 3, below for a screenshot of the interface for creating music in SuperCollider's IDE). While SuperCollider is capable of a great many styles of artistic projects, generative music is a common kind of program written in the language.



```
	SynthDef(\sim, {arg freq=440, out=0, release=1, mg=4;
		var amp = Rand(0.0, 0.5);
		var sig = SinOsc.ar(freq, Rand(0.0, 1.0), amp / mg * 1.2);
		sig = sig * Env.perc(0.0001, releaseTime: release, curve: -8).kr(2);
		sig = CombC.ar(sig, Rand(0.01, 0.4), Rand(0.01, 0.4), Rand(0.01, 0.3), Rand(0.01, 2));
		sig = Pan2.ar(sig, Line.kr(Rand(-1.0, 1.0), Rand(-1.0, 1.0), Rand(0.01, 3.0)));
		Out.ar(out, sig);
	}).add;
	SynthDef(\str, {arg freq=440, amp = 1.0, gate=1, out=0, verb=0.5, release=1, attack=0.01, mg=4, decay=1,
detune=0.01;
		var sig = Mix.fill(12, {
			VarSaw.ar(freq+(freq*Rand(detune.neg, detune)), Rand(0.0, 1.0), Rand(0.0, 1.0),  amp / mg / 10);
		});
		attack = attack*2;
		sig = sig * Env.adsr(attack, decay, 1.0, release * 2).kr(2, gate);
		sig = Pan2.ar(sig, Line.kr(Rand(-1.0, 1.0), Rand(-1.0, 1.0), Rand(0.01, 3.0)));
		Out.ar(out, sig);
	}).add;
	SynthDef(\cha, {arg freq=440, amp = 1.0, gate=1, out=0, verb=0.5, release=1, attack=0.01, mg=4, decay=1,
detune=0.01;
		var sig = WhiteNoise.ar(Rand(0.0, 0.9));
		sig = RLPF.ar(sig, freq, 0.1 + LFDNoise3.ar(1).abs * Rand(0.0, 0.1)) * 0.1 / mg;
		sig = sig.softclip * Env.adsr(attack, decay, 1.0, release).kr(2, gate);
		sig = Pan2.ar(sig, Line.kr(Rand(-1.0, 1.0), Rand(-1.0, 1.0), Rand(0.01, 3.0)));
		Out.ar(out, sig);
	}).add;
	s.sync;
	loop{
		rseq = {Array.fill(num = rrand(2, 12), {arg i; rrand(0, 8)})};
		if(rrand(1, 5)==4, {
			scale = Scale.at(Scale.names.choose);
		});
		if(rrand(1, 5)==4, {
			root = rrand(0, 12);
		});
		Pdef(\main,
			Ppar(Array.fill(rrand(1, 12), {arg i;
				var seq = rseq.value;
				var cursor = rrand(0, durlist.size - 1);
				var durnum = rrand(1, 5);
				var durs = Array.fill(durnum, {
					cursor = cursor + rrand(-2, 2);
					durlist.clipAt(cursor);
				});
				Pbind(
					\instrument, [\sin, \str, \str, \cha].choose,
					\scale, scale,
					\dur, Pseq(Array.fill(seq.size, {durs.choose * time}), inf),
					\degree, Pseq(seq, inf),
					\octave, [2, 3, 4, 5, 6].choose,
					\type, Pfunc({[\note, \note, \rest].choose}),
					\verb, rrand(0.05, 0.8),
					\release, rrand(0.0, 8.0),
					\attack, rrand(0.001, 3.0),
					\root, root,
					\mg, Pbrown(0.1, 6.0, Pfunc({rrand(0.1, 1.0)}), inf),
					\detune, rrand(0.005, 0.015),
					\decay, Pfunc({rrand(0.01, 2.0)}),
```

Search and browse

Search
Search all documents and methods

Browse
Browse all documents by categories

Getting started

These are useful starting points for getting help on SuperCollider:

Getting Started tutorial series
Get started with SuperCollider

Glossary
Glossary

Client vs Server
Explaining the client vs server architecture.

More on Getting Help
How to find more help

All tutorials
Index of all help files categorized under "Tutorials."

Documentation indexes

Documents
Alphabetical index of all documents

Classes
Alphabetical index of all classes

Class Tree
All classes by inheritance tree

Methods

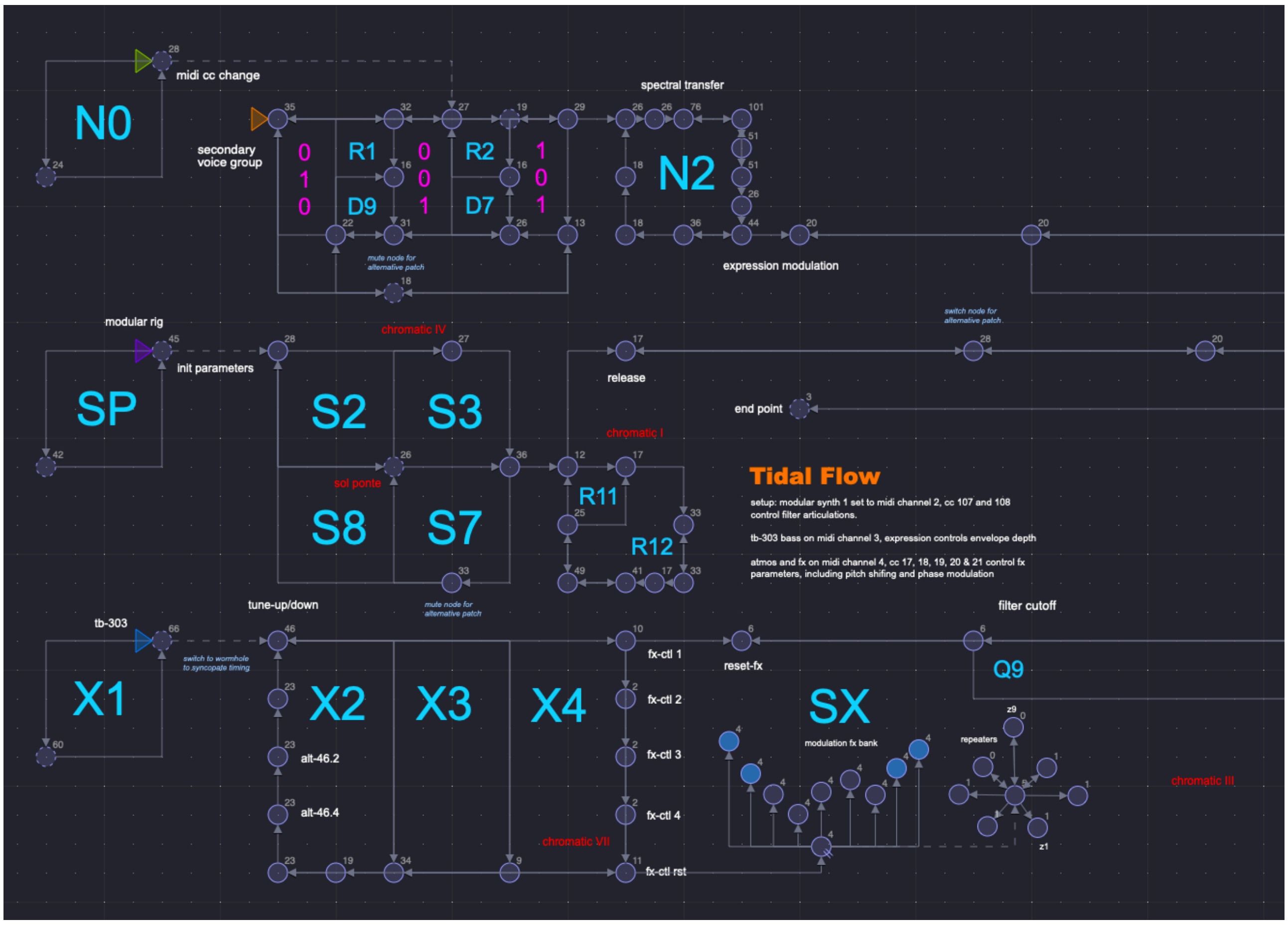
```
	5448 method selectors, 2226 classes
	method table size 13173528 bytes, big table size 97017984
	Number of Symbols 12182
	Byte Code Size 365679
	compiled 323 files in 0.55 seconds
compile done
localhost : setting clientID to 0.
internal : setting clientID to 0.
Class tree inited in 0.01 seconds


*** Welcome to SuperCollider 3.11.2. *** For help press Cmd-D.
file "/Users/declan/Library/Application Support/SuperCollider/Help/scdoc_version" does not exist.
SCDoc: Indexing help-files...
SCDoc: Indexed 1345 documents in 0.6 seconds
```

```
Interpreter: Active  Server:  0.00%  0.00%  0u  0s  0g  0d  0.0dB
```

Figure 3 - A screenshot of the SuperCollider IDE with an example section of generative music code by Zhagun-Linnick (2020)

From figure 3, above, it is clear that while SuperCollider is capable of almost limitless musical creativity on the part of the developer, it is far from approachable when considering a new user. As such, although many of these ideas and styles of creation may be useful in terms of architecture, they were ultimately rejected in this project for a new approach; as the generation was to be driven by, and informative to, any users of the final application, or to any developers who download Processing.

Finally, some examples do exist attempting to be a midpoint between high-complexity solutions such as SuperCollider, or applications in the vein of Eno & Chilvers' work. One such example of that approach is Nodal 2.0, an application that does abstract some of the elements that are generating music for the user, while still retaining a high level of interaction and customisability. However, the interface design is something that is likely too complex and confusing for people to feel quickly comfortable with if they don't have either previous experience with synthesisers, or digital music creation in general.



Tidal Flow

setup: modular synth 1 set to midi channel 2, cc 107 and 108 control filter articulations.

tb-303 bass on midi channel 3, expression controls envelope depth

atmos and fx on midi channel 4, cc 17, 18, 19, 20 & 21 control fx parameters, including pitch shifing and phase modulation

Figure 4 - Nodal 2.0 screenshot from marketing material (SensiLab, 2021)

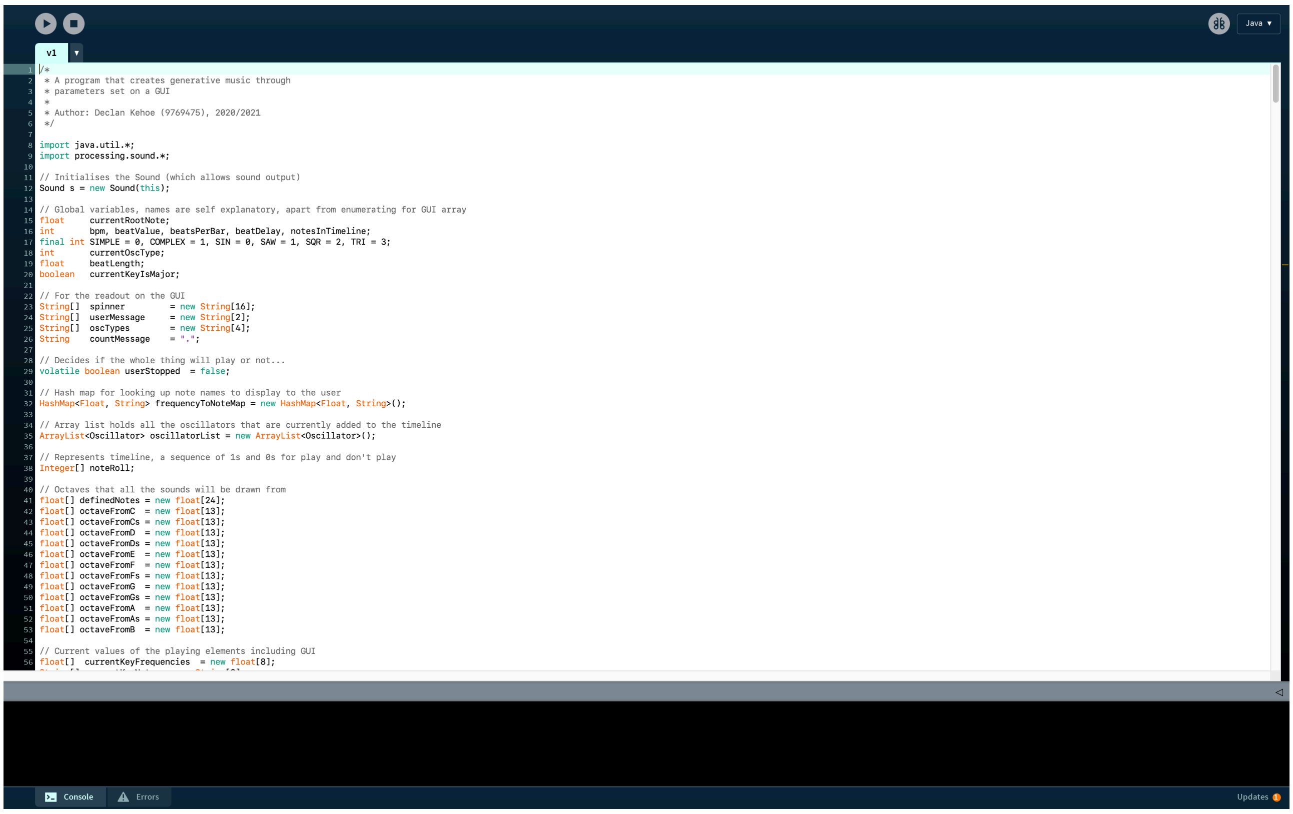
The examples of generative music software described in this section are only a small portion of the wide range of code-based approaches that have been attempted over the decades. While none of them are going to be directly emulated, they have each informed the development and design of this project.

2.3 Processing 3 - Quirks and Considerations

Processing is a project designed more for artists than computer scientists - as such the complexity and level of functionality is sometimes loosely defined, as is the level of abstraction for certain operations (Fry and Reas, 2007). The general style of development found in Processing (and all of the examples found in both the IDE and official website) is to have a single source code file, known as a sketch, that is run from within the Processing IDE. This sketch of Processing Source Code is a .PDE file, which stands for Processing Development Environment and will therefore open in Processing's IDE. Though the language of Processing is functionally almost identical to Java, and the final application runs as a Java application, Processing has its own programme structure and API (Fry and Reas, 2007). There is also an off-shoot of Processing that employs JavaScript for it's code convention and runtime, but that was not used in this project.

A key feature of every processing sketch, is that unlike in Java there is no main method that is always run. Rather, there is a setup method that runs once upon a program starting, followed by a draw method that will continuously loop until the program is exited. This architecture defined most of the technical limitations of the project, but also crucially allowed the flexibility to rapidly prototype and iterate throughout development - giving rise to a fail-fast approach and use of agile methodologies.

Processing was also key to achieving objective 2 (see 1.1) as the final sketch folder could be copied into any Processing developers directory, and run by simply hitting the play button, as seen in figure 5, below. Processing also allows sketches to be exported as compiled Java programs, which greatly aided in the feedback process by allowing non-developer users the opportunity to test the program with a single executable.



```
/*
 * A program that creates generative music through
 * parameters set on a GUI
 *
 * Author: Declan Kehoe (9769475), 2020/2021
 */

import java.util.*;
import processing.sound.*;

// Initialises the Sound (which allows sound output)
Sound s = new Sound(this);

// Global variables, names are self explanatory, apart from enumerating for GUI array
float      currentRootNote;
int        bpm, beatValue, beatsPerBar, beatDelay, notesInTimeline;
final int SIMPLE = 0, COMPLEX = 1, SIN = 0, SAW = 1, SQR = 2, TRI = 3;
int        currentOscType;
float      beatLength;
boolean    currentKeyIsMajor;

// For the readout on the GUI
String[]  spinner         = new String[16];
String[]  userMessage     = new String[2];
String[]  oscTypes        = new String[4];
String    countMessage    = ".";

// Decides if the whole thing will play or not...
volatile boolean userStopped  = false;

// Hash map for looking up note names to display to the user
HashMap<Float, String> frequencyToNoteMap = new HashMap<Float, String>();

// Array list holds all the oscillators that are currently added to the timeline
ArrayList<Oscillator> oscillatorList = new ArrayList<Oscillator>();

// Represents timeline, a sequence of 1s and 0s for play and don't play
Integer[] noteRoll;

// Octaves that all the sounds will be drawn from
float[] definedNotes = new float[24];
float[] octaveFromC  = new float[13];
float[] octaveFromCs = new float[13];
float[] octaveFromD  = new float[13];
float[] octaveFromDs = new float[13];
float[] octaveFromE  = new float[13];
float[] octaveFromF  = new float[13];
float[] octaveFromFs = new float[13];
float[] octaveFromG  = new float[13];
float[] octaveFromGs = new float[13];
float[] octaveFromA  = new float[13];
float[] octaveFromAs = new float[13];
float[] octaveFromB  = new float[13];

// Current values of the playing elements including GUI
float[]  currentKeyFrequencies  = new float[8];
```

Figure 5 - The final project code as seen in the Processing Development Environment. The top leftmost button (⏵) will launch the generative music app as a separate, compiled Java program.

3 - Planning and Development

3.1 Splitting up the task

After early exploratory work getting to grips with the specifics of working with Processing, the decision was made to split the design and development into three separate but related strands, or subsystems, that could be worked on mostly independently. The subsystems consisted of:

1. Sound - which incorporates both the generation and manipulation of the applications sounds. This would make extensive use of Processing's Sound library.
2. 'Generative engine' - a term used in this project to describe the rules driving the sound generation, the tracking of the app's state, and the changes in the quality of the sounds being made. Overall this subsystem was unquestionably the most demanding technical challenge, and required careful balancing of complexity and ease-of-use.
3. GUI - the interactive interface used to play or stop the app's sound production, to influence the parameters of the generative engine, and to optionally view the current rules playing out. Multiple iterations of the GUI were developed, then discarded or refined, with most changes directly based on user feedback.

While all three subsystems had to be integrated at different stages, the use of sprints to complete thin slices of functionality within each subsystem gave multiple benefits. Specifically, it allowed familiarisation with Processing itself, an opportunity to break things quickly to learn how they work, and to continuously bug find and iterate; all before needing to worry about the specifics involved in developing the overall application as a monolithic entity. This approach also had the benefit of meaning the subsystems were largely modular, and could be tweaked extensively before increasing the overall instability or complexity.

3.2 The Core Design

Fundamentally the three subsystems of the design process are reflected in the eventual functionality of the app. While the GUI, generative engine, and sounds are continuously interacting, they each perform their functionality somewhat in isolation, only passing relevant data to other processes as required - in keeping with generally regarded good software design practises. This subsection consists of a high-level description of the overall design and how the subsystems interact, with the design and operation of each subsystem explained in greater detail separately.

The sounds are all generated from objects and methods found in Processing's Sound library, which includes many tools for sound making and manipulation. For this project, the objects used were Oscillators, Envelopes, and a single Sound object. Each of these are described in more detail in the Sound section of the report (4). The sounds are all controlled by the generative engine (see section 5 for more on this).

The engine that serves at the heart of the application is best understood as a timeline; 16 equally-spaced points in time that each determine if a sound is heard, or silently moved past. The timeline has its pattern of sounds and silences continuously cycled through (similarly to the tape loops described in section 2.1) until either the user stops it, or completely quits the application. While analogous to a beat sequencer in other music creation software or hardware, the timeline is never directly interacted with by the user, rather, the parameters and timing of the sounds or silences are indirectly influenced by the user's actions. The user's perception of the timeline, dynamically editing it (including during playback), how best to represent the engine's status, and how to ensure that all subsystems could accurately and swiftly interact and change; all while still being under the asynchronous control of the user, was by far the largest technical challenge of the project. It also needed to be communicated clearly and unobtrusively to the user.

The GUI represents all of the parts of the program relevant to the user, and can be toggled back-and-forth between two levels of complexity by clicking in the 'readout window' at the the top of the app. Large interactive buttons are used for user input, and the readout window is used for almost all user feedback. The timeline is the only element of the GUI that is present at all times and in all modes, demonstrating the centrality of the timeline to the functioning of the app (see section 6 for more on the GUI).

3.3 Project Timeline

Multiple iterations were developed throughout the project's lifetime, and many of these were used by people with various experience of both music creation software, and comfort with technology generally. Feedback from these users was then incorporated into the next cycle of development. A timeline is shown in fig. 1, below, and each named milestone is described in greater detail after the figure. The Sound, Generative Engine, and GUI strands each have sections dedicated to their development (sections 4, 5, and 6, respectively.)

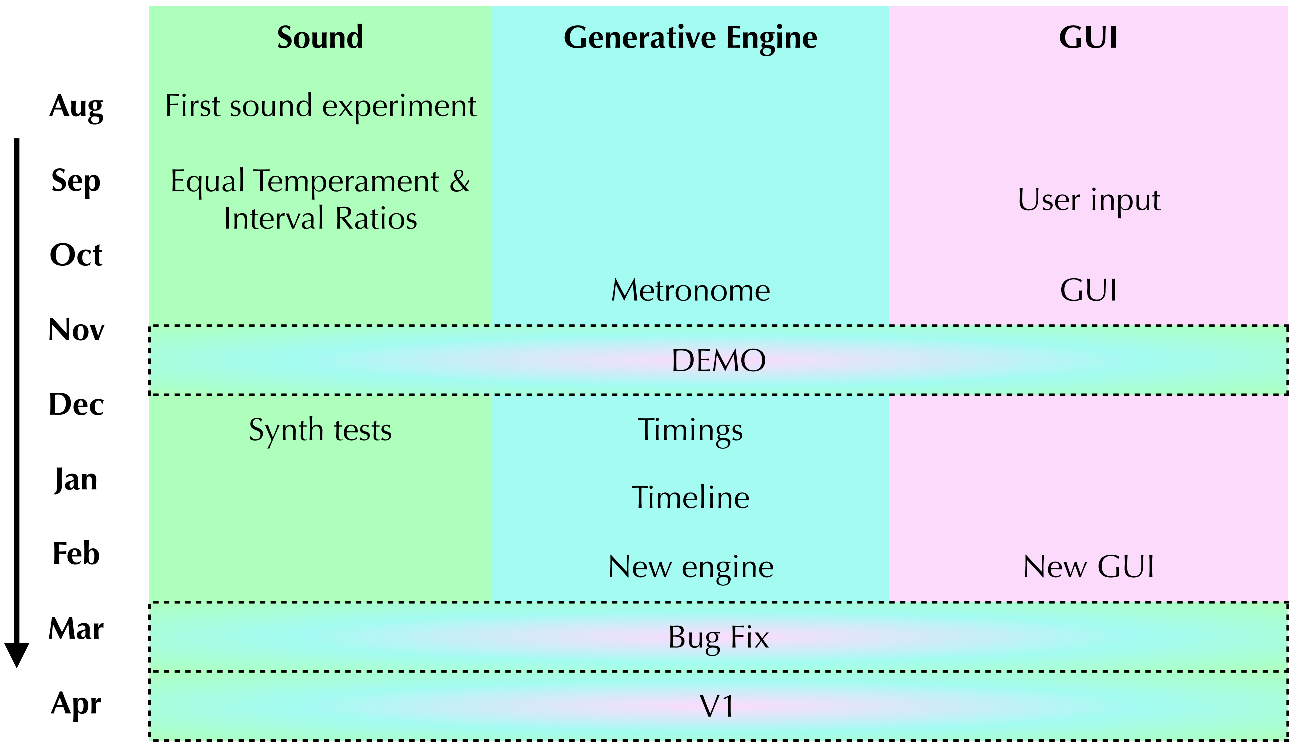


Figure 6 - Timeline of development milestones from August 2020 to April 2021

3.4 Milestones

First Sound Experiment - An initial investigation to understand the basic functioning of Processing's setup and draw structure, as well as first hearing and interacting with Processing's Sound library and Oscillators. This initial experiment was based on the example code for the Sound library (Processing, 2020a). It featured frequency and amplitude settings, and mapped their values to the mouse's position on screen.

Equal Temperament & Interval Ratios - Research was conducted into how ratios can be used as a way to define the frequency differences between notes in a scale (see 4.1 for more on this topic). This was not used in the end but the research was informative. The research into the intervals between notes that are the basis of modern pitch was the eventual definition used for note frequencies in the final application. These milestones marked the first time note frequencies were defined in the setup as discreet variables that were then used by calls to Processing's Oscillator class.

Attempts to marry the temperament and interval ratio ideas were also ultimately abandoned as they eventually lead to dissonance, but there were very early prototypes in these milestones where interactivity from the user changed the notes. Each key press of an appropriate letter on the keyboard would use the corresponding note as a root note (again, see 4.1 for more on this topic). From the root notes either a minor or major chord would play, which could be changed by clicking the mouse in the program window. The vertical position of the mouse in the program changed the volume of the sound that was playing.

User Input - The first in depth experimentation where the user could tweak parameters. Not a true GUI in a meaningful sense as there was no feedback other than the sounds produced. This version allowed the user to define the root note (as was the case in Intervals), but it also allowed a choice of the type of chord triad that would play by using the up/down/left/right keys on the keyboard to change to a major/minor/augmented/diminished chord respectively. Also this was the first time where the user could stop sounds without needing to quit the entire app!

Metronome - A very early version of the generative engine, and an attempt to get a representation of the beats per minute (bpm), or speed at which the application progressed. This was very buggy, and most of this attempt was scrapped and redeveloped. It did, however, teach a great deal about the mechanics of Processing's internal clock, and how the refresh rate of the draw cycle is tied to that. It also informed how future definitions of the bpm would function.

GUI - Quite simply, the first iteration of an actual GUI. This was an important milestone in that it was the first chance to begin iterating on the user-facing aspects of the application.

Demo - The first integration of the three development strands. This milestone featured a slightly more polished version of the GUI which crucially contained the first iteration of the readout screen. There was still no use of a timeline, or pattern looping, and the note timing was tied to the oscillators themselves. Many of these ideas would be rebuilt wholesale, but this early milestone was the first that could be considered a real 'application'.

Synth Tests - An initial test of the different types of wave produced by the classes of Processing Oscillator, which in turn produce different synthesiser sounds. This milestone - based on the demo - was when the limitations and instability from tying generation to the oscillators became more apparent.

Timings - A crucial milestone where the generative logic was completely detached from the oscillators, and the first instance where the oscillators ran in their own threads.

Timeline - The first milestone where the final design was almost completely realised. Not only superficially in the readout, button layout, and functions of the GUI, but also in attempting to control the decoupled logic and oscillators asynchronously with a timeline.

New GUI - The final functionality and iteration for the final version of the GUI. Milestones of the GUI would be minor iterations going forward.

New Engine - The final version of the logic and the architecture of the links between the generative engine, the oscillators, and the GUI. From this version on, iterations were for polishing, bug fixing, and minor refinements.

Bug Fix - A bug fixing pass from the previous milestone which was used as a second demo version, or beta, for the final version that would be developed for the project.

V1 - More polish, and a final bug related to removing oscillators identified in the previous demo milestone, Bug Fix, was fixed. This is the final version developed for the project, and is the version that participants who were interviewed for the evaluation used (section 7).

4 - Sound Subsystem

4.1 Sound, Ratios, and Scales

Research for the sound system began with an exploration into relevant music theory. Specifically into the division of note frequencies into scales, and then how to get the oscillators to use this in a way that was aurally pleasant. This research also informed how to eventually represent the values that comprised the notes and scales that would be implemented.

For the sake of understanding the eventual implementation of sounds in the project, a level of familiarity with music theory is necessary. Please note, a glossary is available for many terms, and an overview of the relevant sound and music theories explored for this project is presented in this section.

Sounds are, fundamentally, vibrations. The frequency (speed) of a vibration dictates a sound's relative pitch, sometimes described from low to high. The amplitude (height), of the vibration dictates the volume (loudness) of the sound. The frequency and amplitude are required to define what is heard by someone hearing a sound, and the time it takes for a sound to change between frequencies and amplitudes is what will give it a specific quality. A combination of related frequencies from a sound source create harmonics, and produce the characteristics or timbre that distinctly define an instrument or noise.

The nature of sound dictates the qualities in music that make it generally considered pleasant, harmonious, or dissonant. In western music, the possible vibrations a sound can make are divided into 12, and noted by a letter and symbol, in a pattern which repeats every time the vibration doubles. The notation typically used is:

A♮ | A♯/B♭ | B♮ | C♮ | C♯/D♭ | D♮ | D♯/E♭ | E♮ | F♮ | F♯/G♭ | G♮ | G♯/A♭

after which the pattern repeats. In practise, a vibration of 440Hz is called A(4), while 880Hz is A(5). A complete cycle of the pattern of 12 notes is called an octave, and ordering a subset - most commonly 7 - of the notes from within an octave by the size of the gaps between them, called an interval, is a scale. A piece of western music will typically play notes from only one scale, which is referred to as the piece's key. Finally, playing different notes simultaneously creates a chord, where the interval between the notes that are playing dictates the kind of chord that is played.

An initial idea for the project was to have all pitches in a given instance of creating music generate automatically based on a starting note. This idea can work in principal, and could be seen as an elegant way to define the intervals between notes. Historically, the ancient Greek mathematician Pythagoras and his followers are said to have developed a theory of music based on ratios between note frequencies (Crocker, 1963). The basis of this 'Pythagorean-tuning' is that a frequency difference between two notes at a ratio of 3:2 is the most pleasing to the human ear. While certainly pleasant, issues arise when trying to work with more than a few notes at a time, as sticking strictly to a 3:2 ratio will soon lead to frequencies that are not defined as notes, and music is made up of more than perfect 3:2 ratios. While Pythagorean tuning uses mathematical techniques and the nature of live music to get around some issues of dissonance, the rigidity with which a computer driven oscillator will stay at a given frequency means tracking and correcting for any dissonance was counter productive. Given this, after initial experimentation and research (3.4), using this approach was dropped in favour of using predefined values for each note that would be available to play.

Throughout history, music theory has used multiple versions of different ratios to define the frequencies of notes in a scale between the ends of an octave. Unfortunately, there is no perfect arithmetic way to subdivide the frequencies that compromise an octave into an even scale across multiple pitches. Therefore, western music has generally settled on dividing an octave into 12 parts which are equally spaced on a logarithmic scale, with every interval being a step of 1/12 of the octave it is in (Temperley, 2007). By then using seven of these twelve equal-temperament notes in an octave, a scale can be established. This standard is used for the frequency values of all notes and scales in this project.

4.2 Final Note Implementation

During the setup phase, twenty-four note frequencies (two octaves) are defined and assigned to global float variables. From these, a nested for-loop creates arrays of each octave from a different root note, which are themselves global variables. While it would be possible to instantiate these arrays of octaves at the point they are required - and discard them when no longer necessary - the relative simplicity afforded by having them always available was appreciated during development. It was also in line with a project objective (1.1). A map is also built at this point of Floats to Strings so that the value of a note can be used to look up its related letter notation. This map is used to display on the detailed readout the note values of the current key the application is using to generate music.

The notes that can be used to generate music at any given time are defined by the key that the application is working in at that point. This group of notes, known as the current key notes, are an array that is emptied and filled each time the key is set, or reset. To fill the array, an interim array is set equal to one of the already made octave-from-*X* arrays, which is determined randomly. From this array, there is an equal chance of either a minor or major version of a scale from that root note being loaded into the current key note array. For a major scale it is the 1st, 3rd, 5th, 6th, 8th, 10th, and 12th note frequency values of the array, and for minor the 1st, 3rd, 4th, 6th, 8th, 9th, and 11th. These notes in the array represent the respective intervals for the kind of scale that is being used for generation.

4.3 Waves and Envelopes

The other key aspect of sound that was examined for the project, was how the frequencies generated by the oscillators themselves would sound. Initially only a sine wave oscillator was used, but Processing also includes square wave, triangle wave, and saw wave oscillator types. Each of these waveforms produces a distinct quality of sound, which is something that was wanted as a way to make the sounds produced by the application more interesting to the user. As such, a method to track and specify the currently desired oscillator type was needed.

There was also the need for consideration of envelopes. In the context of sound synthesis, an envelope refers to the way the sound produced goes from silent to its maximum volume, holds at that maximum, and then returns to silence. There are third party libraries for extensive modification of envelopes, but one of the goals of this project was to use only native libraries if possible (1.1). Fortunately, there is an Envelope class which can accept envelope parameters in the native Processing Sound library.

The envelope settings included with Processing allow for an amplitude relative to the maximum value to be defined from 0.0 - 1.0, as well as the length of time (in ms) for the sound to reach, hold, and then fade from that amplitude. There are various types of envelope for synthesisers with various levels of granularity for specifying length of time and amplitudes at different points throughout the sound's duration. But the envelope available in Processing is known as an ASR, (or Attack, Sustain, Release) envelope.

Attack defines how long it takes the oscillator to reach its maximum amplitude. Sustain defines how long the oscillator will stay at that amplitude, and is also used by Processing to denote what that maximum amplitude is. Release defines how long it takes the oscillator to fade from the maximum amplitude back to silence.

By combining the various types of Oscillator (sine, saw, square, triangle) and different sets of ASR envelope parameters to effect an Envelope controlling each Oscillator's output, a large range of tonal qualities can be created by the application. In the interest of preserving the goal of simplicity for the user (1.1), there is no way for them to directly set these parameters. Rather, there is an ability to change to a new synthesiser type which requires changing to a different oscillator waveform, and every note added to the timeline has a semi-random set of values applied to its ASR envelope.

The reasoning behind the ASR envelope's parameters being semi-random is simply that truly random values would mean a greater chance of generating unpleasant or barely audible sounds; something that would directly contradict the comfort and empowerment of the user. The specifics of how the amplitude and timings are assigned is discussed further in the next section on the generative engine (5).

4.4 The Oscillator Method

After multiple iterations it made most sense to develop a class that had a constructor which combined the parameters for the frequency, amplitude and wave form of an Oscillator, and the attack, sustain, and release amplitude and timing for an Envelope. While named simply Oscillators, the objects constructed by this class function as a combination of Processing's existing Oscillator and Envelope objects. This class was made after an earlier integration with the generative engine used to create new oscillators directly, but which had several weaknesses in terms of stability, and difficulty in controlling oscillators after they were instantiated.

The current design of the sound in the application also means that there is no longer a single, or single set, of oscillators being used throughout the lifetime of the program with parameters being constantly changed. Instead, each new note added to the timeline represents an entirely new instance of an Oscillator object with an unchanging set of oscillator and envelope parameters, that will function in its own thread until being terminated.

The constructor for a new Oscillator requires a number of parameters. Specifically they are:

- givenOscillator - a Processing Oscillator object, which will be part of this custom oscillator class
- oscType - an integer from 1 to 4 indicating the kind of oscillator that is to be created. Where 1 = a sin wave, 2 = a saw wave, 3 = a triangle wave, and 4 = a square wave type.

- givenNote - the frequency of the oscillator, as a float, which will equate to the pitch of the sound produced (see 4.1 for more)
- givenAmplitude - the amplitude of the oscillator as a float, which will equate to the volume of the sound produced (see 4.1 for more)
- givenEnvelope - a processing Envelope object, which is attached to the oscillator in order to shape the sound it produces, and requires values for its attack, sustain, and release parameters (see 4.3 for more)
- givenAttackTime - how long, in milliseconds, the attack phase of the envelope will be (see 4.3 for more)
- givenSustainTime - how long, in milliseconds, the sustain phase of the envelope will be (see 4.3 for more)
- givenSustainLevel - how large, as a float, the amplitude of the sustain phase of the envelope will be (see 4.3 for more)
- givenReleaseTime - how long, in milliseconds, the release phase of the envelope will be (see 4.3 for more)

With all of these parameters, the custom Oscillator class object can be constructed, and will then be added to the global oscillator list in order to be triggered by the timeline, as described in section 5.2.

This class also allows for oscillators in the oscillator list to quickly be replaced by reusing many values. For example, iterating over the list and replacing each individual oscillator with a new one where only the type of waveform, or the note frequency, changed is partially the process which enables the functionality described in section 5.3.

5 - Generative Engine

5.1 Early Architecture

The first attempt to incorporate generation tied the elements of randomisation and timing to the oscillators themselves (3.4). In practise this meant that within the main draw loop of the program (2.3), each oscillator was already instantiated. During playback, a new thread was started for each oscillator, within which parameter changes to the oscillators or their envelopes would be applied at the end of their respective loops. The user had three buttons to click;

- A 'Generate!' button which started the playback thread.
- A 'Stop' button which exited the playback thread and stopped the overall sound output of the app.
- A 'Randomise Key' button which changed the pool of notes from which the randomiser setting the oscillators parameters would draw from.

This early iteration did at least have a successful generative element; at the end of each loop of playback there was a randomiser giving an equal chance for every oscillator to keep their current parameters, or change to new, randomly generated ones. In this way, the application creating an ever-changing cycle of notes was achieved, but there were many significant issues that arose from the implementation.

The main drawback arising from this approach was that the oscillators would become unresponsive to any instructions until completion of their current playback cycle. This also had the secondary effect of putting the entire application in a blocked state after clicking to change the key, and before the oscillators had accepted these new options. Finally, the other key issue was that the length of each playback loop - both the sound itself, and the silence before it started again - was controlled by the oscillator's envelope timings and a delay parameter, all of which were attached to the oscillator in the playback thread.

Overall, while the early method did work to an extent in creating looping and evolving sounds, there was no in-app way for a user to alter the rules behind the generation, or to choose the amount of sound generating elements. To alter the generation, a developer would have to quit the application completely and edit the source code. Given the limitations, and the failure to meet some of the stated objectives (1.1), a rethink of the approach to generation was required. This rethink resulted in the development of the timeline.

5.2 Creating the Timeline

Threads in Processing are a built in part of the language, however they don't allow for much in the way of customisation, control checks, or safe closure. They are implemented as a method, which is passed a String stating the name of another method, which will then be run in the new thread. There are no other Processing methods for interacting with the threads, and any further functionality with threads would require importing and extending Java's Thread class. This would be against the second project objective (1.1), so the app would be required to use threads as they were already implemented in Processing. Processing's documentation recommends that the best use of threads is for them to be implemented in processes with a definitive program flow with a predefined beginning and end (Processing, 2020b). This meant that the initial attempt at a generative engine using a single thread to continuously loop through the oscillators, change their parameters, and wait for their respective playback cycles to complete was not a good use of the thread method.

Given the limitations of tying the timing and parameters to the oscillators directly in a single looping playback thread, the architecture of the generative engine was fundamentally changed. This change was the timeline. Now, instead of changes to the sound being passed from the main draw loop to a playback loop which, in turn updated each oscillator after they had completed their current sequence; a timeline on the main draw loop holds a sequence of 1s and 0s to indicate if an oscillator should play, or there should be no new sound, respectively. The draw loop then acts as a 'ticker', which moves through this sequence of 1s and 0s, one beat at a time, at a speed dictated by the current beats per minute of the engine (see 5.3 for more detail). For example, a timeline of [0, 0, 1, 0, 1] would have the ticker move past the first two points without playing anything, an oscillator would be triggered at the third point, no new sound on the fourth, and a second oscillator would play at the fifth. The timeline in the application has 16 points, and therefore a maximum of 16 oscillators can be triggered to play in one loop of the timeline. Going forward, 1s are referred to as 'notes' or 'oscillators' contextually.

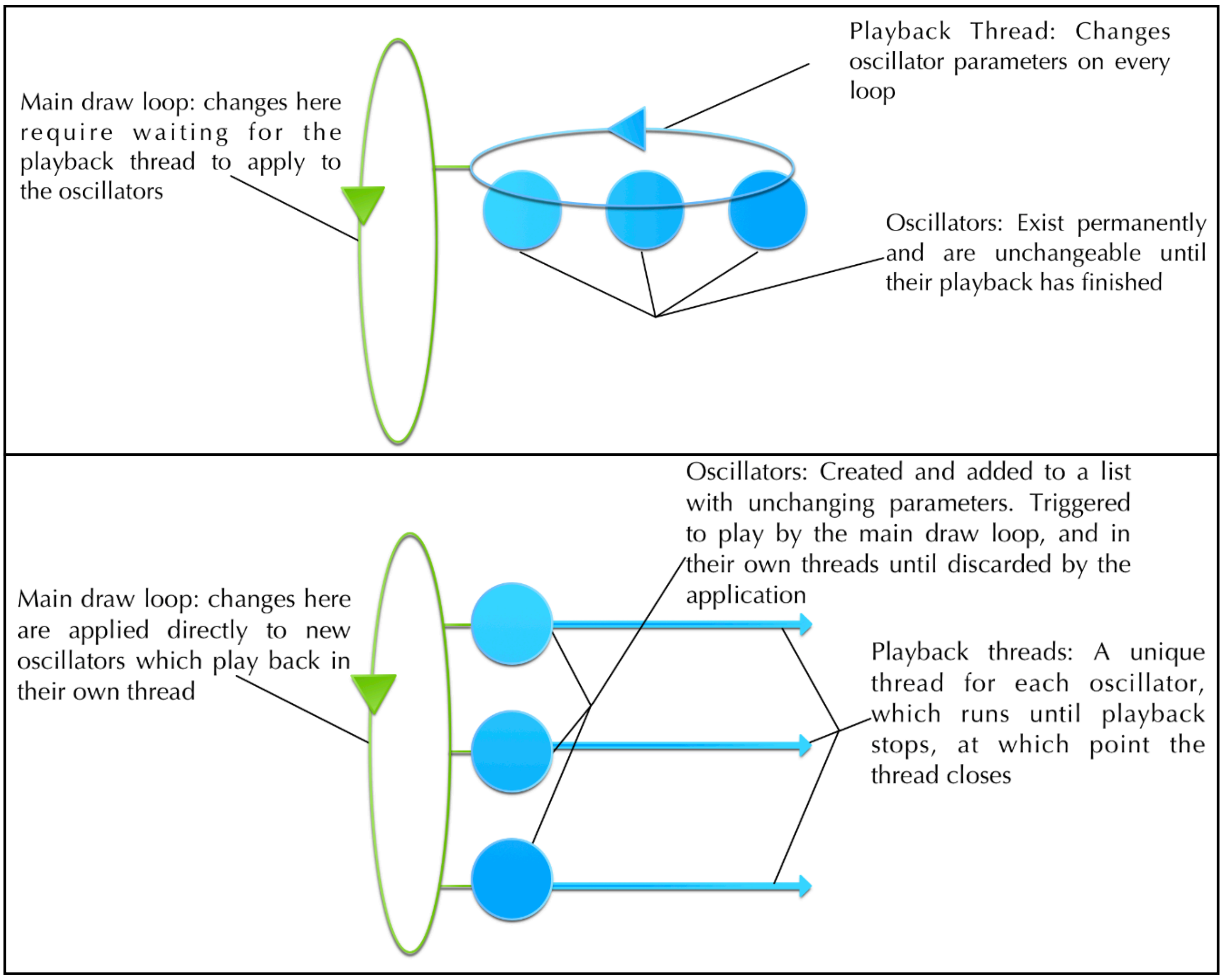

Playback Thread: Changes oscillator parameters on every loop
Main draw loop: changes here require waiting for the playback thread to apply to the oscillators
Oscillators: Exist permanently and are unchangeable until their playback has finished
Oscillators: Created and added to a list with unchanging parameters. Triggered to play by the main draw loop, and in their own threads until discarded by the application
Main draw loop: changes here are applied directly to new oscillators which play back in their own thread
Playback threads: A unique thread for each oscillator, which runs until playback stops, at which point the thread closes

Figure 7 - Original application architecture above. Final application architecture below.

This system requires that for every 1 added to the timeline, an oscillator with randomly generated parameters is added to a list, and as the timeline progresses one oscillator after another in this list will play, each in their own respective thread. This implementation means that the list of oscillators can be edited even as the ticker progresses through the timeline, such that oscillators can be removed and replaced at any time.

A list is required to handle the oscillators as it allows each of them to be discreetly replaced with new ones, all with their own parameters as required, and without interfering with any other part of the program. This is in contrast to the original method which could be very unstable and required attempting to modify a fixed set of oscillators throughout the runtime of the application. Please see figure 7 above for diagrams visualising the two versions of the engine architecture.

This engine change meant that the timeline was always running, and therefore always able to be interacted with, as well as allowing a wider range of parameters to be set asynchronously to the oscillators. For example, allowing the timeline to create new oscillators in their own threads meant the user could add - or remove - an arbitrary number of oscillators at will, or change the kind of wave the oscillators produced. Crucially, the change also fixed the issue whereby the user could not stop the oscillators from blocking changes until they had completed their individual playback.

Functionally, the timeline is an array of 16 integers that are either 0 or 1, where 0 represents silence, and 1 represents that a sound should be played. This array (named noteRoll in the code) is what the main draw loop iterates over, calling Oscillators to play.

While the limitation of Processing's threads mean that there is no way to directly monitor, alter, or gracefully exit during oscillator playback - the final implementation has been robust enough to satisfy the objectives of this project (1.1).

5.3 Finalising Functionality

Once the generative engine architecture of using the timeline to hold parameters for oscillators was established, the final range of functions available to the user were able to be finalised. Given that lowering the barrier to entry for new users was required for objective 3 (see 1.1 for more specifics), a choice was made to limit the potential for a user to make 'bad' choices. That is, choices that would result in discordant or overly-complex sounding generated music. This choice meant strictly defining for the user a set of boundaries within which the generation would be allowed to work, as well as the amount of elements that would be generating simultaneously and shown to the user. In other words, while the design of the timeline meant an arbitrary number of oscillators could be playing simultaneously, the overall sound and interface was designed with the aim of keeping the user engaged and comfortable, and to feel in control of what is occurring with the application throughout its runtime. The interviews in section 7 are used to analyse if this aim was in fact met.

The functions that could be interacted with by the user through the GUI are; Add a note, Remove a note, Change Key & Tempo, Change Sound, Play, and Stop. Each button when pressed will trigger a method defined in the program, and in this sense the generative engine's parameters can be updated by the user. Each of these functions are discussed in more detail below.

Add a note - clicking the button will add a note to a space on the timeline that does not currently have one. To function correctly this method must first ensure that the timeline is not already full - which is established with a global variable tracking the amount of notes in the timeline up to the maximum of 16 - and will alert the user if this is the case and exit the method. If the timeline is not full, a randomiser will generate an integer between 1 and 16 to place the new note in the array, a check then takes place to ensure that slot is free. If free, a 1 is placed at that point, and if not the randomiser generates a new integer between 1 and 16 and the check is performed again. The check and regeneration happens repeatedly until a blank slot is found. While this may be somewhat inefficient when compared to a hashmap or similar, it ensures two things; that adding notes to the timeline always results in a random placement and distribution of sounds, helping with the mutative elements of generative music, and that non-native library use is avoided, in line with the project objectives (1.1). The variable tracking the total amount of notes in the timeline is updated at this stage. Adding a note to the timeline also requires adding an oscillator to the oscillator list. The parameters for the oscillator are described in more detail in section 4.4)

Remove a note - this works similarly to adding a note but in reverse. A check is performed to ensure the timeline is not empty, and if it is, the user is alerted. If there are notes in the timeline one is removed randomly using an inversion of the process of adding a note. Once a note has been removed, the global variable tracking the total number of notes in the timeline is updated, and the first oscillator in the oscillator list is removed.

Change Key & Tempo - there are global variables instantiated in the setup phase of the program which are used to track the values of the key and tempo, and these global variables are what dictate multiple aspects of the program at any point. The key determines the pool of notes which are possible to assign to an oscillator, and the tempo is used to define both the speed of the ticker (as described in 5.2), as well as the possible values of the attack, sustain, and release timings of an envelope (described in 4.4).

Whenever this button is pressed, the global variables for the key and tempo are changed. For the key, this is achieved by simply generating a random number to use as the index from the array of root notes. A second random element is then if the key will be major or minor (see 4.1 for more) which dictates the full list of notes able to be assigned to an oscillator. The tempo is defined with an algorithm that gives a range of values which are multiples of 5, and between 40 and 120, which will be the beats per minute of the current generation. These bpm are then converted in to a 'beat' defined in milliseconds, which is used as the base unit for defining the timings of the envelope, as well as the speed of the ticker.

Finally, the list of oscillators is iterated over, and every oscillator is replaced with a new one with values which conform to those dictated by the new current key and tempo, and keeping any other parameters not related to key or tempo.

Change Sound - this button will first cause the program to generate a random number between 1 and 4, which will be used to denote which of the sound wave types (saw, sin, square, and triangle, respectively) the application will use its oscillators to produce. To achieve this, the oscillator class I developed uses the number to instantiate the required kind of Oscillator object from Processing's sound library; each of which uniquely produce a single waveform shape.

The current wave type is tracked with a global variable, and the random number which is its value will be regenerated until it is different to whatever value it had before the button was pressed. Once the number is different, the list of oscillators will be iterated over, and each will be replaced with newly instantiated oscillators copying existing key and tempo settings, but will have whatever waveform type corresponds to the new random number.

Play - once clicked, the method attached to this button will begin in a new thread, and a volatile Boolean tracking the state of the app as either playing or not playing is updated. It will reset the global variable tracking which number in the oscillator list to play next, as well as the ticker's position on the timeline, to zero. The ticker will then move forward by one place and check to see if that new position it is at on the timeline contains an indication to play, or remain silent. If there is a 1, indicating a sound should be played, the oscillator in the list matching the number to play next is started in its own thread, and the number to play next is increased by one. After delaying by whatever the current beat length is, as determined in the key & tempo section, the ticker moves forward another place, and the checking process begins again.

The main complexity of this method comes from both updating the user message for the complex version of the GUI with all of the current information of what is being played (see 6.2 for more detail), and by checking to ensure that the number for the next oscillator in the list to play actually exists (see 6.4 for more).

The information to display to users is derived from the global variables tracking key and tempo. The tempo is displayed via the bpm, and the notes in they key are from converting the float value of the precise frequency of each note in the current key, into the letter representing that note. The conversion uses a Float to String dictionary, built in the setup phase, which contains all the defined note's frequencies as keys, and names as values.

Stop - this button first checks that playback is currently occurring using a global volatile Boolean. If it is, then it iterates over the list of oscillators, and uses their stop playing method to stop them playing. There is one key weakness in this implementation, which is discussed in the next subsection, 5.4.

5.4 Errors and Weaknesses

Overall, the final version of the app is very stable, and none of the users interviewed for the final evaluation reported crashes or errors (see section 7). This is mostly achieved through a mix of attempting to anticipate potentially dangerous behaviour, such as adding more notes that can fit on the timeline, or removing notes once they are already gone.

The main fragile point in the application is most likely the threads which enable the asynchronous interaction with the generative engine during playback, and the volatile Boolean that is used to indicate whether the play thread is currently running or not. Again, careful handling and tracking of the state of this Boolean, and ensuring that setting the value of it is handled atomically by the play and stop methods have ensured that in the final version of the app, no errors have been detected or reported.

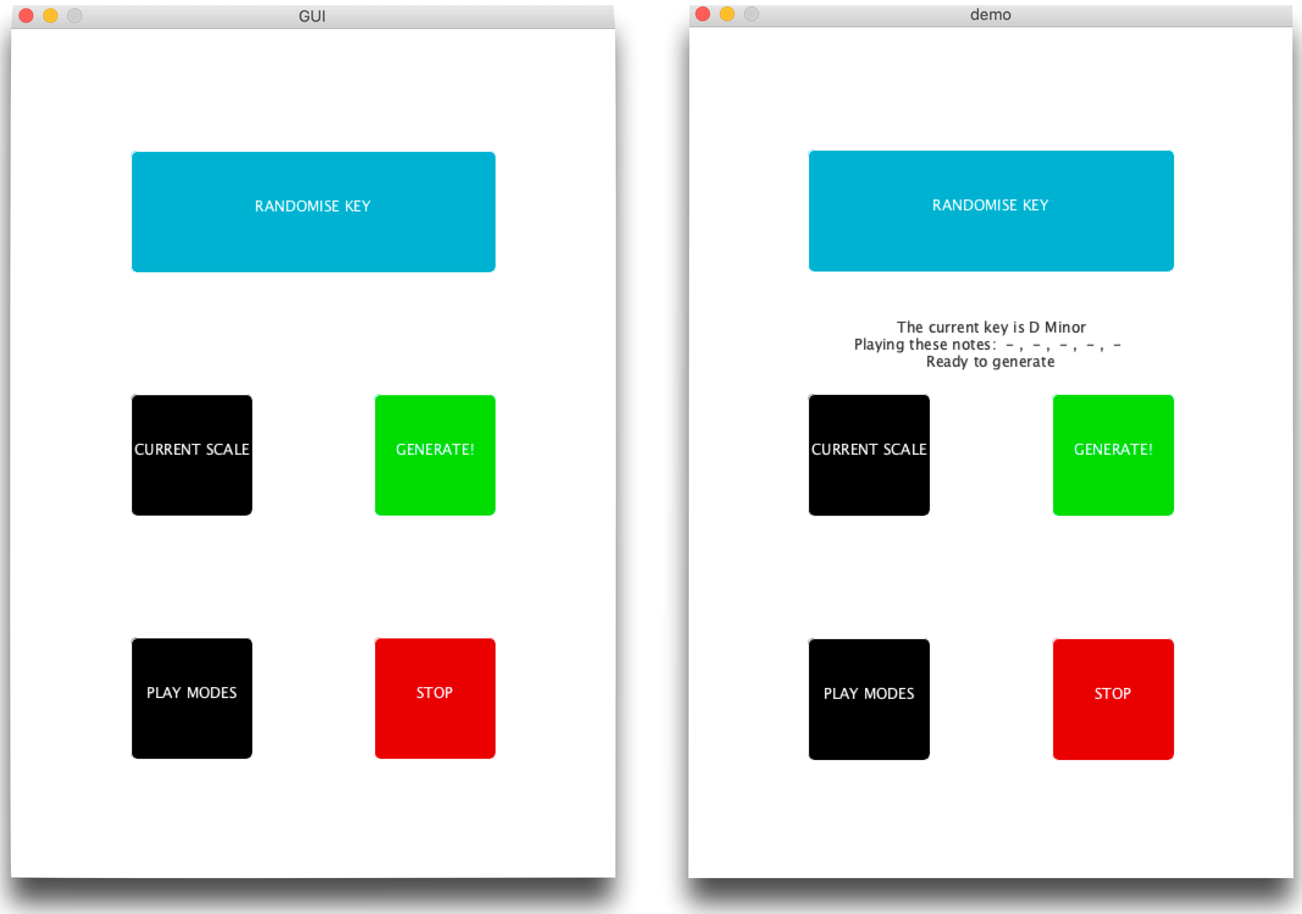
Another error that was avoided was a potential index out of bounds error, which could have occurred without careful management of the next oscillator to play number, used during playback. If the user were to remove notes from the timeline during playback, and before the ticker has reached the end of its current loop of the timeline, the number for the next oscillator to play could be greater than the actual number of oscillators in the list. Therefore, during each tick of the timeline, a check is made to ensure that the number to play next is not higher than the size of the list, and setting that next to play number to 0 if that is the case.

A final weakness to highlight is one that can occur when the stop button is pressed after having recently used the ‘change key & tempo’ or ‘change sound’ options. Due to the way these methods are implemented, the oscillators in the oscillator list which have the old key, tempo, or sound are replaced by new ones. If the old oscillators are currently playing, they will continue to make a sound until they have finished their complete playback cycle and their threads are terminated. However, due to the fact they are no longer in the list of oscillators, it is not possible to use their stop playing methods to immediately cut off the sound they are making. The solution in the current build is to display a message to the user whenever the stop button is pressed which informs them that any excess sounds will fade. This was specifically cited as reassuring by at least one user (see 7.3), and so is an acceptable fix. However, a more robust way of stopping sounds would be preferable. Perhaps having a secondary list to track recently removed oscillators before their thread has ended would be useful, but ensuring that this then didn’t become overcrowded and excessively memory-heavy would become another issue to tackle.

6 - GUI

6.1 Prototyping

One of the main benefits of Processing is the ability to rapidly build and test a complete application, rather than requiring a large amount of infrastructure work to be done before seeing the results. This was used a great deal throughout the process of creating the application for this project, especially regarding the iterations that were developed for the GUI. Initially there was no specific idea for the look of the application, other than the guidance provided by the objectives (see 1.1) that required a non-intimidating, welcoming experience for the user, regardless of their previous experience with similar software.

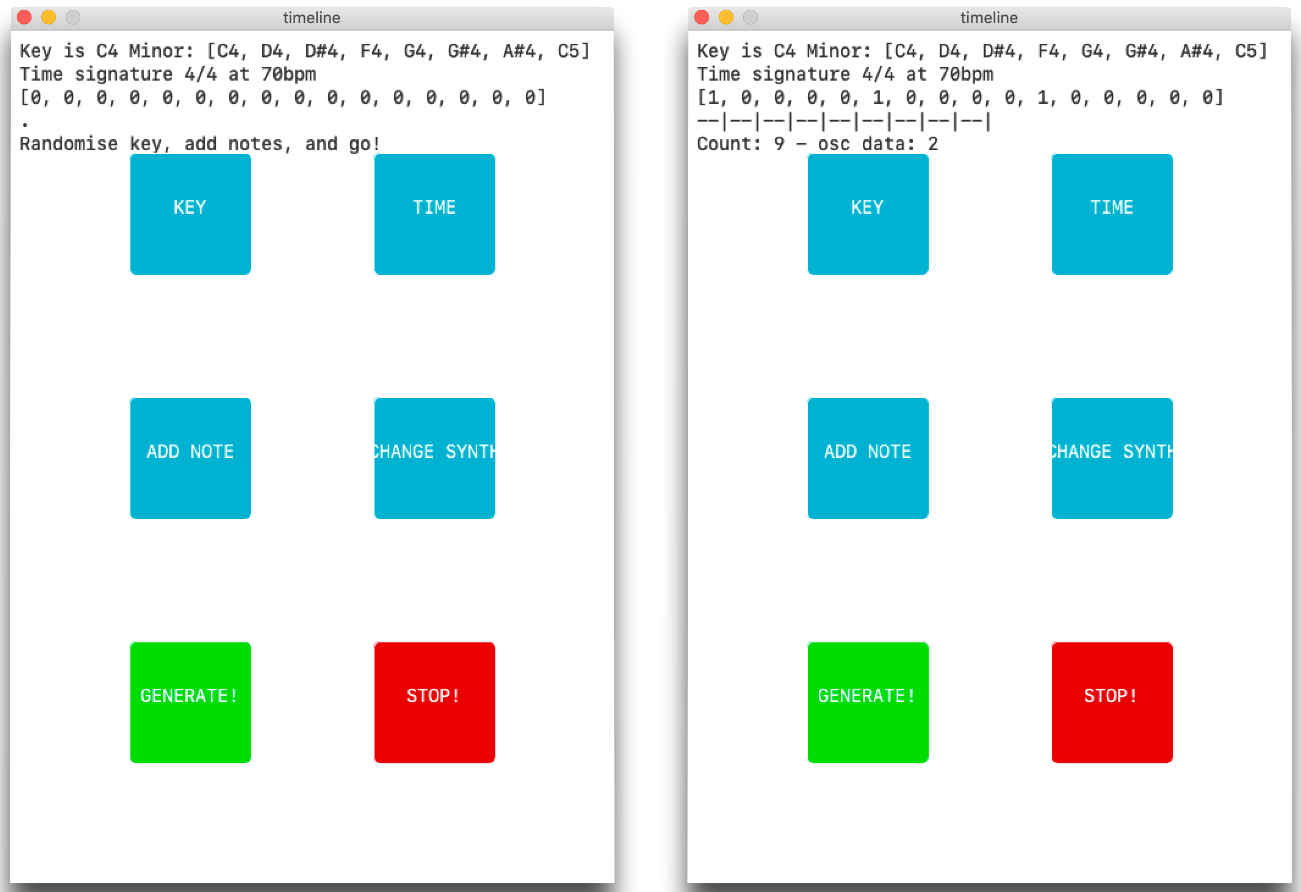


The current key is D Minor
Playing these notes: - , - , - , - , -
Ready to generate

Figure 8 - The first iterations of the GUI. Left shows how there was no readout originally, with all feedback to the user being only the sounds they heard. Right shows the first version of the readout, but the placement and functionality of the buttons is far from completed.

A key requirement was made for simplicity, and for usability, both of which are principles of good UX. The initial versions of the GUI are superficially similar to the final version, however there are some striking aspects that are missing, or otherwise malformed. There was initially no concept of a readout to directly tell the user what was being used to dictate the rules of the current generation, and the functionality that was offered was also different. As can be seen in figure 8, the most prominent button was used to randomise the current key. There was also colour coding to show what would start and stop the generation process. However, the wording of the labels on the buttons was ambiguous to those who were not already familiar with the functioning of the app, the layout did not conform to any particular recognisable standard, and the readout, when it was added, was positioned in a horizontally centrally location, but between buttons.

Overall, while there was certainly a great deal of useful experience gained from rapidly prototyping these early GUIs - particularly in terms of the use of Processing's draw loop to track the mouse and to drive the program - a huge amount of refinement of the GUI was required. This refinement wouldn't be fully possible until the final amount of interactivity the user had with the app was decided.

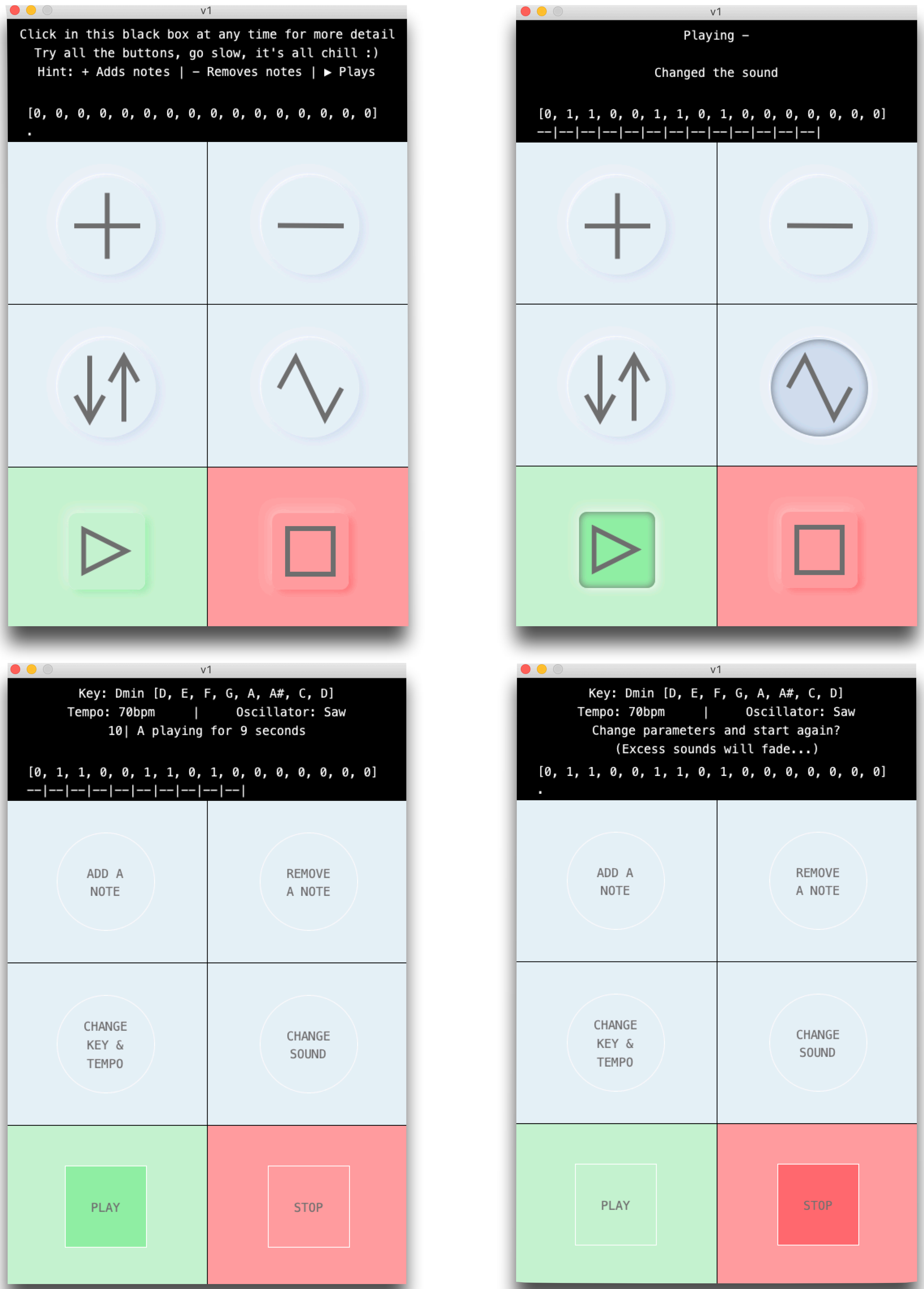


```
Key is C4 Minor: [C4, D4, D#4, F4, G4, G#4, A#4, C5]
Time signature 4/4 at 70bpm
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
.
Randomise key, add notes, and go!
```

```
Key is C4 Minor: [C4, D4, D#4, F4, G4, G#4, A#4, C5]
Time signature 4/4 at 70bpm
[1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]
--|--|--|--|--|--|--|--|
Count: 9 - osc data: 2
```

Figure 9 - Midpoint of the project GUI, note the presence of the timeline and ticker

Around the middle of the development time for the project, the timeline was introduced. This became an anchoring point from which the rest of the GUI was based. Figure 9 (above) shows how the addition of the timeline prompted a complete rethink of the readout's position for the user, and more importantly the kind of information that would be shown. Specifically, the display of the timeline and the 'ticker' underneath it - which moves whenever the application is generating to show its current position in the timeline - were vital references for the all future iterations of the GUI. At this point, there was no option to remove notes, rather, when clicking the 'STOP!' button all points in the timeline would return to 0. While this behaviour was later changed due to a judgment that it could frustrate users if the notes they had added were removed because they had stopped playback, in one of the evaluative interviews (see transcript) a participant actually did expect this behaviour, and was actually surprised to see the notes remaining in the timeline after pressing stop!



```
Click in this black box at any time for more detail
Try all the buttons, go slow, it's all chill :)
Hint: + Adds notes | - Removes notes | ▶ Plays

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
.
```

```
Playing -

Changed the sound

[0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0]
--|--|--|--|--|--|--|--|--|--|--|--|--|
```

```
Key: Dmin [D, E, F, G, A, A#, C, D]
Tempo: 70bpm      |      Oscillator: Saw
10| A playing for 9 seconds

[0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0]
--|--|--|--|--|--|--|--|--|--|
```

```
Key: Dmin [D, E, F, G, A, A#, C, D]
Tempo: 70bpm      |      Oscillator: Saw
Change parameters and start again?
(Excess sounds will fade...)
[0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0]
.
```

Figure 10 - The final version of the GUI. Top left shows the simple mode screen upon first opening the app, top right shows the simple mode having pressed the 'change sound' button during playback, bottom left shows the complex mode during playback, and bottom right shows the complex mode after pressing 'stop'

6.2 Final Implementation

The final iterations of the design were established when the functionality of the application was settled on, as well as the decision to have two modes for the GUI. This decision was made so that there was a simple interface immediately upon opening the app that would not overwhelm any novice users, but easy access to a more detailed version to meet the objective of informing more curious users (1.1).

Instead of using Processing to prototype, the final version was designed first in Sketch, and an interactive prototype was made to mockup a user's potential journey through the app. This mockup was also used to define a colour scheme that was pleasant while having interactions still be readable, and to design the glyphs that would represent the various available functions in the simple mode (see figures 11 & 12).

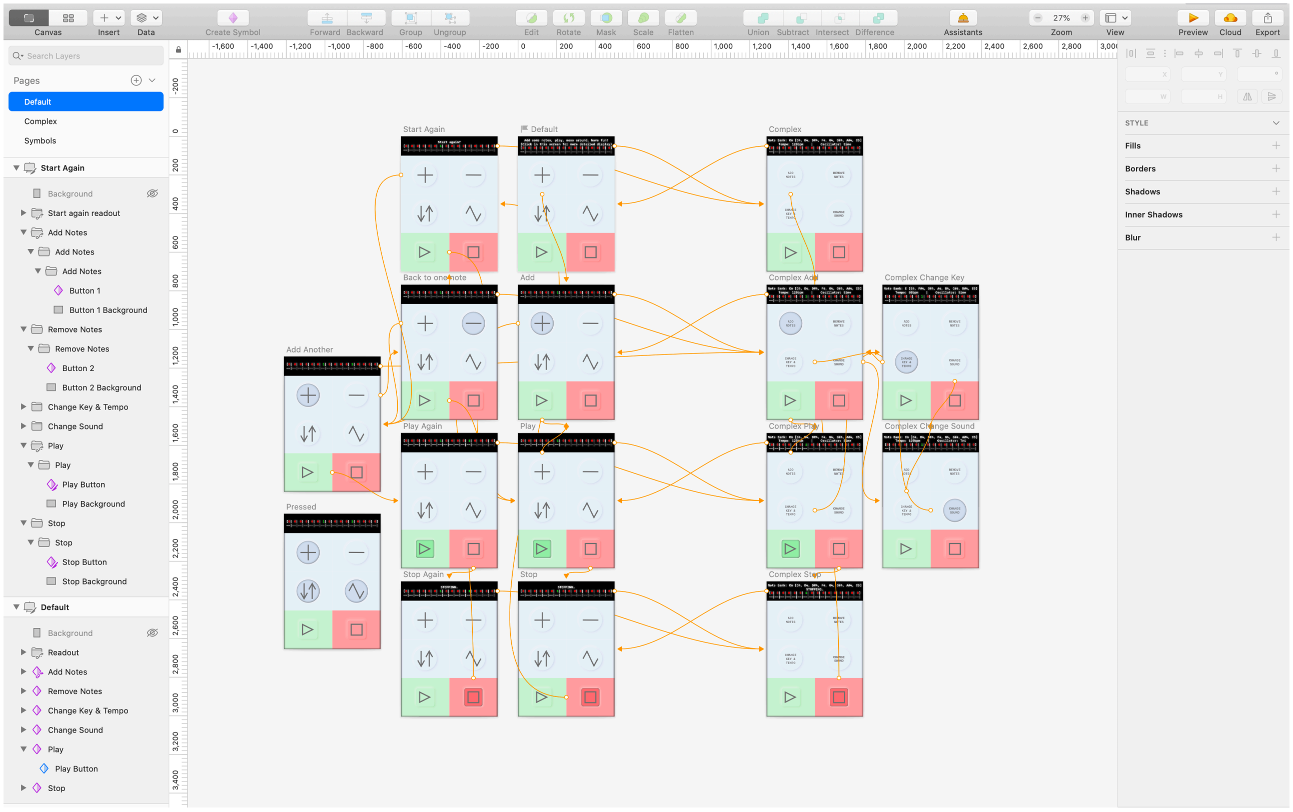


Figure 11 - Screenshot of the Sketch app showing the layout for the interactive demo

While almost all of the design elements were translated from the mockup to the application in Processing, the nature of the IDE meant that there was a level of interpretation to be done to achieve the correct look. Each constituent shape comprising the elements of the GUI (circular buttons, rectangular readout, square button-surrounds) were placed by a method which defined their colour, location within the app window, and either the text or glyph displayed on them. Processing's draw loop enables the responsiveness of the GUI, so that buttons change colour when the mouse hovers over them, and indicate when they have been pressed.

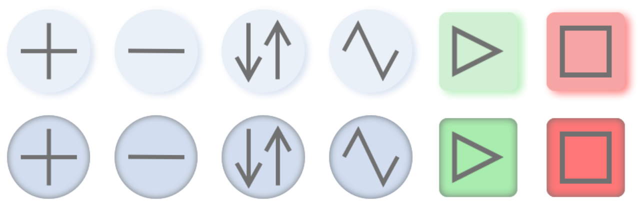


Figure 12 - The un-pressed and pressed versions of the glyphs created for the app

The readout area at the top of the screen updates constantly based on the current status of the application, and can be toggled between the simple and complex modes by clicking anywhere within the edges of its box. The readout section is also used to provide updates and reassurance to the user through text, and a conversational style was chosen for these messages to further the sense of being welcoming, and to reduce intimidation.

The timeline and ticker from previous iterations is always present. Key, tempo, and waveform information is visible if in the complex mode, as well as labels on the buttons to explicitly declare their function.

Overall I am personally pleased with the eventual look and feel of the GUI, and was impressed with Processing's flexibility to both allow the level of customisation to the developer, while still remaining relatively simple to implement. Further evaluation based on user experiences is found in the evaluation section of the report (7).

6.3 Limitations

One limitation of this method was that the values for placement are absolute, and not relative. This means that the app can only ever be run in a separate window at a fixed size, and any resizing would have to be done by changing the hardcoded values in the source code.

The most important limitation by far though, is the lack of accessibility options. This is related to issues with resizing described above - in that if a user cannot see the text at its current scale, there is no way to increase it - but also that there is no screenreader compatibility. Further, there is no way to change the colours of the GUI, which may be an issue for those who are colour blind, or who require a higher contrast in their displays. All of these issues are things I would like to address in future updates to the application if there was more time, and would be the main priority if that application was then to be distributed more broadly.

7 - Evaluation

7.1 Semi-Structured Interview Questions

A series of participants were asked 7 questions on their experience of using the application after using it for around ten minutes. The questions were designed to conform to the qualitative ideals of getting useful information from them (Braun and Clarke, 2006).

An attempt was made to interview users with a range of experience levels of both music creation and digital audio applications, as well as general technology proficiency and comfort. While the majority of participants interviewed had not used music creation software, or created music in the real-world, their answers still provided valuable insight into the successes and failures of the project in relation to its initial goals (1.1).

The questions are listed below, please note that there are two versions of question 7 which are asked depending on the specifics of the pre-interview app usage situation. If the participant was not observed using the app then the first is used, and if they were observed, the second.

1. What previous experience (if any) do you have with digital music creation?
2. What previous experience (if any) do you have with real-world music creation?
3. What was your impression of the app upon first opening it?
4. How intimidating was the application's functionality to find, understand, and use?
5. What (if anything) frustrated you about using the app, or didn't work as you expected?
6. What (if anything) did you enjoy about using the app?
7. Did you spend time with the more detailed readout, and if so, what did you learn from it?
    - Or, if the participant has been observed using the application:
7. If using the app more, how much time do you think you would spend on the more detailed readout, and, did you learn anything from it when you did use it?

7.2 Key Themes from Interviews

Selected quotes from the interviews are used throughout this section to illustrate some key themes identified of the strengths and limitations about the user experiences with the app. Please note that the appendix contains complete transcripts of all the interviews, and provide much greater context for each participant's answers, and therefore a more complete understanding of their experience and feedback.

Simplicity and ease-of-use

> *"The enjoyment of being able to create a sound, then created a secondary interest in me, to look and see what those sounds meant…"*
>
> — Participant S

Participant S had no previous experience of either digital or real-world music creation, so their interview provided a good indication that the stated goal of being welcoming to more novice users was met, in that they were able to find enjoyment in using the application to make sounds. This was a sentiment shared in other interviews, with all other participant's specifically responding to the question of what (if anything) they had enjoyed about the app referencing either the actual sounds that were generated, or feeling a sense of accomplishment in creating them.

The positive reactions to the application were often rooted in the sense of simplicity found in the interaction style of the app. Participant E put their feelings towards the simplicity of the UX thusly:

> *"I got the green play, and the red stop straight away. And then I thought, 'oh gosh, I wonder what these other buttons do?' So, I just started clicking on them *laugh*! And then pressed play, and then thought 'wow, it makes a sound!' So, I though it was really… the way it looks, I like it, it's really simple, it's really neat."*
>
> — Participant E

Even for users with experience of using both real-life and digital music creation tools, such as participant N, the simplicity was regarded as a positive. In their experience the simplicity was not removing a barrier to entry - their previous experience of digital audio workstations (DAWs) meant this was unnecessary - rather, it was a way to start making sounds quickly:

> *"…with the app itself, you can open it and in two seconds you've got something!"*
>
> — Participant N

Beyond discussion of the app itself, participant N was the only one interviewed who also chose to look at the code after using the app. This was a helpful insight into the potential success of objective 2 (1.1), which includes the aim of allowing other people to easily pick up and extend the code.

> *"I did have a look at the code after - but not having had a look before opening it, it was, illuminating, I guess. You could see quite a bit of what was going on, which was very interesting."*
>
> — Participant N

Participant N's experience seems like an ideal representation of the objective, their use of the app was simple, and informative, then as a more curious user they were able to take the information gleaned from using the app to then move to the code that made it work. Within the code itself they could recognise how the code and final app interacted.

Confusion with aspects of playback or UX

Two main UX issues were made apparent throughout the interviews. One was an occasional disconnect between an expected outcome and actual app performance. But the other more complex issue related to toggling between the simple and complex readout modes.

The issue of disconnect between expected outcome and actual experiences was articulated clearly in two passages from Participant's K & S, respectively.

> *"I think that I was expecting it to be… quicker. Like, the change of the sounds to be quicker.*
>
> …
>
> *My expectation was that I would hear the change immediately, but it didn't happen."*
>
> — Participant K

> *"…an interpretation problem for myself, was it took a while to grasp that, when I stopped it, that that didn't mean it automatically returned to 0."*
>
> — Participant S

In the case of Participant K, the issue was primarily based around the lag between adding an oscillator to the timeline, and then actually hearing it. They noted that this may be a part of "the nature of generative music", but the fact that this wasn't communicated is a weakness in the UX design of the application.

Similarly, Participant S's expectation that stopping playback would cause the timeline to be reset to 0 is interestingly a behaviour similar to what was in the application at a previous stage of development. In this previous version (see sections 5 & 6 for specific detail), there was no remove note button and the concept of the timeline was very different. Users would choose to add and play up to five oscillators simultaneously, with an option to stop all the oscillators and start from 0. This interaction style was replaced by the add/remove paradigm which seems to have been preferable for many users, but evidently not all.

The toggling of the simple and complex readout modes is a more nuanced aspect of the feedback to understand. This functionality is an attempt to meet the third project objective (see 1.1), that the application can optionally give "information to curious users as to how the generative aspects function in the context of music theory generally".

Mixed responses to use of the detailed readout

The interviews presented a range of responses to the readout modes. While some participants appreciated the ability to choose their own level of detail and the relative insights they offered, others either didn't appear to see a benefit in the ability to switch, or would have missed the functionality completely without external guidance.

There was a general division in the decision to use the detailed readout more or less based on the relative previous interest the participant had in music creation. Specifically, the more interested a participant had been previously, the more they said they would use the detailed screen to learn about either the details of their generative music, or to use the sounds they had heard and apply them to other music. For example, participant N, who had the most real-world and digital music creation experience, described that they would use the detailed mode; not necessarily for the music theory information, which they already understood, but for the apps functionality more technologically

> *"I'd spend most of my time in the advanced mode. I did learn quite a bit of how it [the application] worked."*
>
> — Participant N

Participants K & E, who had some previous real-world music experience, both explained that they were more likely to use the detailed screen, in order to better understand some of the music theory behind the generation. Participants P & M, however, who had no previous related experience, said they would mostly continue using the simple version:

> *"I'd use it [the detailed screen] some. But I'd use the one with the plus and minus, the more simple layout. I don't need to overcomplicate it because I know how it works now."*
>
> — Participant M

Relatedly, participant S, who also did not have previous music experience, said the detailed readout had actually made them curious to learn more:

> *"[the detailed readout] made an interest for me to see, I don't know, maybe a glimpse into a world I don't know anything about."*
>
> — Participant S

7.3 Specific Improvements from Interviews

Some useful and specific constructive criticism arose from the interviews. Although there is no ability to incorporate the suggestions in this section into the application in a way that can be demonstrated, any further iterations of the app that would be developed outside of the assessment period would certainly take them on board.

These kind of suggestions are also indicative of the development process, where the ability to export the project from Processing into a standalone application and share with users to gain their feedback, was used to rapidly iterate and improve the app experience. A representative example from this final set of more structured interviews and questions is:

> *"If I was to change anything, it would probably be that instead of clicking in the black box, it'd maybe be a button there instead to change over to the next screen."*
>
> — Participant P

This is quite frankly a brilliant suggestion, and something that would definitely be a useful inclusion in a further iteration of the GUI! It would solve the problem of a user missing their chance to see the message informing them of the more detailed mode, which is definitely a weakness in the current version of the app. And the presence of a button incentivises clicking the area, while also reassuring the user that they are not going to jeopardise the stability of the app.

Another specific issue was mentioned by Participant E, they said:

> *"…when I pressed the stop button and it didn't stop, I wasn't irritated but I was like 'oh my god!' I thought I'd broken it!"*
>
> — Participant E

While Participant E went on to say that the message (as described in 5.4), explaining that excessive noises would fade did reassure them that the noises they were hearing did not mean the app was not working correctly, the fact there was a moment of panic is problematic. A potential solution would be to have a more robust way of ensuring that all noises were stopped as soon as the stop button was pressed. However, implementing this in such a way as to ensure that the transition between oscillator sound types, or key and tempo changes would be difficult, as the fade out from delisted oscillators is part of the more friendly UX design the application aims to produce. The current solution, while not ideal, is at least stable and reassuring enough to not knock the overall confidence of the app user.

7.4 Self-Reflection

Basing this self-reflection on the originally stated objectives seems the best way of contextualising the successes and failures of the project from my own - inevitably somewhat subjective - viewpoint. This section will go through the original objectives one at a time, and contrast them to their status in the project in reality.

1. A functional application must be built, that can be used to create generative music.

I believe this has largely been achieved. There is inarguably an application, and given that it has been stable for multiple users I also think it can be fairly described as functional. To ascertain the goal of creating generative music, I think the definition of generative music is important. To me, I believe this is somewhat mixed in terms of success.

The application can produce sounds that are based on a set of rules, and those rules are not predetermined by a composer directly, so in that sense I believe the idea of creating generative music has been accomplished. However, if a user were to leave the application completely after initiating a sequence, there would eventually be a point where the cycle could no longer be considered 'ever-changing', which is another quality often used to define generative music. As the oscillator's playtimes continue to overlap and interplay, eventually a meta-cycle would repeat, so without some form of user intervention the self-contained nature of many generative music systems would not be present.

A potential improvement to more fully meet this first criteria could therefore be some way of keeping random mutations of the oscillators found in the timeline. This would, however, be a non-trivial change. Many more novice users would likely be confused and frustrated to have changes occur without themselves creating or driving them, and presenting the mutation as an optional mode could incur an increase of complexity that would go against objective 3.

2. *A minimal use of non-native libraries should be used for development, to better allow new Processing developers to understand and extend the code, and to personally develop a better grasp of Processing's capabilities.*

Non-native libraries were not used in this project. In fact, the only imported libraries are Processing's included Sound library, and Java's standard array utilities. Both of these libraries will be present on a processing developer's computer without any specific configuration; Processing's own libraries are always present, and built-in Java libraries will be present as Java itself is a prerequisite for developing with Processing.

To ensure the code would be extensible to further developers, the success I believe is again due to both the nature of the code itself, and to the design of Processing. The code was heavily commented, so that someone with little to no understanding of the application should be able to quickly understand its design. The design of processing means that the sketch folder is all that is required for another developer to continue working. In fact, aside from the .png's that were created for some of the GUI elements, a single .pde file would be all that a new Processing user would need to be able to compile and work on the application from source.

Finally, my own understanding of Processing was absolutely improved; I went from having quite literally never used it, to developing this application. The fact that the syntax is entirely Java-based was a massive help in being able to quickly understand the fundamentals of writing for Processing, but the nature of the setup and draw program flow did take some getting used to. Overall, it was a truly useful learning experience, and gave me a real appreciation for the very elegant design decisions that the Processing team took when creating the IDE.

3. *The user experience should not be intimidating, instead welcoming users regardless of their previous experience with generative music, music composition, or music creation applications generally. It should also optionally provide information to more curious users as to how the generative aspects function in the context of music theory more generally.*

This is by far the most subjective aspect to evaluate, and the majority of my opinion is based on the participant interviews above (7.2). Given the feedback from other users, and from the informal feedback and personally-lead changes throughout the many iterations leading up to the final version, I think the project has on the whole met this objective. Specifically, I believe that not intimidating users and making them feel welcome to experiment has been fully achieved, although there is no UX that is perfect or without room for improvement.

To further the topic of what can be improved, the change between simple and complex modes was a solution to the problem of informing more curious users, but by definition, this does increase the complexity found in the application. For some users, this was welcome, but for others it was less-than-ideal. As such, further iterations and feedback sessions would undoubtedly mean a more pleasing solution could be found for a greater variety of users.

There is also the possibility of allowing further functionality to be accessible. While I believe that this would go beyond the remit of an application attempting to be approachable and not intimidating, it would certainly raise the ceiling of what the application was capable of performing for more experienced or creation-interested users.

8 - Conclusion

This report has attempted to systematically describe and explain the process of my project's lifetime. By first reviewing the motivation and objectives of the project, and comparing the proposed application to contemporary applications, a context to understand the rest of the report was established.

Next, the planning and development design was explained, including a summary of the overall timeline and various milestones that were achieved during that time. Following this, a section each was devoted to the three subsystems that comprised the main design pillars of the application, in order to fully explain the decisions behind their respective designs, and the eventual mechanisms behind their functionality.

Finally, the application that resulted from the described process was evaluated in a section that contained both personal reflection, and selected extracts from a series of semi-structured interviews with multiple app users.

It is my sincere hope that this document is a useful and thorough witness to the development work that was completed throughout the academic year. Although the application produced could without doubt be improved with more time, more iteration, and more functionality; I believe that it represents an artefact that meets the initially stated objectives, and is in its own way, a fun piece of software engineering for an end user. The fact that some of those interviewed expressed a sense of accomplishment and enjoyment from using the app was incredibly gratifying, and the experience of creating something almost completely from scratch in an unfamiliar language and development environment was both challenging, and rewarding.

As mentioned in the introduction, I encourage the reader of this report to have some hands-on time with the application themselves. But if they cannot, I hope that this report was an acceptable substitute for the experience of creating their own generative music, or at least helped them to learn more about it.

Glossary

BPM - Beats Per Minute. How many times a beat of the current length would play within a minute, this relates to the ‘rhythm’ of a piece of music, or the relative speed with which the music appears to flow

GUI - Graphical User Interface

IDE - Interactive Development Environment

Amplitude - The height of the waves that make up a sound, which equates to the relative loudness or volume of that sound. A higher amplitude means a sound is perceived as louder.

Frequency - The speed with which a sound wave oscillates between its high and low points, with each speed meaning a different note is perceived by the listener. Related to pitch.

Pitch - The note that is heard by the listener. Can be relative (as in higher or lower pitch to another note), or absolute (the pitch of this note is 440Hz, which is an A, or A4 specifically).

Oscillator - A device which creates a movement at a certain frequency. In software terms it moves from 0 to 1 and back again at a specified speed, and this can be used to drive speakers which will then produce the pitch associated with that frequency

Envelope - A way to control the relative time a sound spends at a given amplitude at various points throughout its playback. The envelope characteristics of a sound is one of the main components which give it a unique timbre

Timbre - The quality of a sound, that gives it a unique character. For example, the same note being played by a guitar and a piano is differentiated by the timbre of those instruments

Attack - The period of time at the start of a sound which goes from an amplitude of 0 to a defined amount

Sustain - The period of time at which a sound is held in the middle of it’s playback, or the level of amplitude at which it is held

Decay - The period of time at the end of a sound playing back where the amplitude goes from the level it was at during the sustain phase, and returns back to 0

Octave - a complete cycle of the 12 notes making up the notes in western music

Scale - a group of notes from within an octave that are defined by the gaps between them

References

Braun, V., Clarke, V. (2006) 'Using thematic analysis in psychology', *Qualitative Research in Psychology*, Vol. 3, No. 2, pp. 77–101 [online] Available at: https://www.researchgate.net/publication/235356393_Using_thematic_analysis_in_psychology (Accessed: April 2021)

Crocker, R. L. (1963) 'Pythagorean Mathematics and Music', *The Journal of Aesthetics and Art Criticism*, Vol. 22, No. 2, p. 189 [online] Available at: https://www.jstor.org/stable/427754?seq=1 (Accessed: April 2021)

Eno, B. (1996) 'Generative Music', *In Motion Magazine* [online] Available at: https://inmotionmagazine.com/eno1.html (Accessed: April 2021)

Eno, B., Chilvers, P. (2007) Trope [application] available at http://generativemusic.com/trope.html (Accessed: December 2020)

Fry, B., Reas, C. (2007) 'Processing Overview' [online] Available at: https://processing.org/tutorials/overview/ (Accessed: October 2020)

Herremans, D., Chuan, C.-H., Chew, E. (2017) 'A Functional Taxonomy of Music Generation Systems', *ACM computing surveys*, Vol. 50, No.), pp. 1–30 [online] Available at: https://arxiv.org/abs/1812.04186 (Accessed: April 2021)

Parviainen, T. (2017) 'How Generative Music Works - A Perspective′ [online] Available at: https://teropa.info/loop/#/title (Accessed: November 2020)

Processing. (2020a) 'Sound' [online] Available at: https://processing.org/reference/libraries/sound/Sound.html (Accessed: September 2020)

Processing. (2020b) 'Thread' [online] Available at: https://processing.org/reference/thread_.html (Accessed: September 2020)

SensiLab. (2021) 'Nodal 2.0' [online] Available at: https://nodalmusic.com (Accessed: September 2020)

Temperley, N. (2007). 'Tuning and temperament'. *Encyclopedia Britannica* [online] Available at: https://www.britannica.com/art/tuning-and-temperament. (Accessed: April 2021)

Zhagun-Linnick, A. (2020) ′Boreas' [source code] Available at: https://sccode.org/1-5d1 (Accessed: April 2021)

Appendix

Transcripts of the interviews conducted for the project are included in this appendix.

The researcher (denoted as Declan) asked the questions described in section 7.1 after around ten minutes of application use by the participant.

All interviews were conducted remotely due to the COVID-19 pandemic. As such, a mixture of screen-sharing-with-audio and audio-only interviews were conducted, dependent on the available compatible technologies.

When participants were not complete strangers to the researcher, a reassurance of objectivity was confirmed by both parties before conducting each interview.

Permission was granted by each participant for a transcript to be made of the interview, and shared in this report.

Interview 1 Transcript - Participant P

Declan: What previous experience (if any) do you have with digital music creation?

P: Zero.

Declan: What previous experience (if any) do you have with real-world music creation?

P: Zero.

Declan: What was your impression of the app upon first opening it?

P: Since I don't really know anything about what any of the notes would actually do, or sounds that they would make… confusing.

Declan: Confusing, that's fair. In terms of the design, the visual design, how about that?

P: The visual design makes a lot of sense… it's fairly straightforward with everything that you'd be looking for. If I was to change anything, it would probably be that instead of clicking in the black box, it'd maybe be a button there instead to change over to the next screen.

Declan: How intimidating was the application's functionality to find, understand, and use?

P: Once you kinda get to the next screen, and you kinda start seeing the different parts of it, it makes a lot more sense.

Declan: So would you rather have seen the other screen first?

P: I think… that… in the fact that I was able to kinda get the hang of it in like, less than two minutes. You can… it's pretty straightforward then, how it works.

Declan: Do you mean the second screen, or how it already is?

P: Both of them. So, with flicking in between the screens it's… it's very straightforward in the sense of, like, what you can do. But it looks like it has a lot of depth to it, to actually figure out what you would like everything to sound like.

Declan: What (if anything) frustrated you about using the app, or didn't work as you were expecting?

P: I suppose it would be just on, moving on to the next screen. That, when things get going, if the message doesn't pop up anywhere after, to click on the black screen again.

Declan: Like you lost your chance to see it?

P: Yeah.

Declan: What (if anything) did you enjoy about using the app?

P: Seems pretty fun *laughs*! The sounds are interesting, when you turn the volume down a little bit…

Declan: *laughs* If using the app more, how much time do you think you would spend on the more detailed readout, and, did you learn anything from it when you did use it?

P: A detailed read out of instructions?

Declan: No, the more detailed screen, did you learn anything from that, that you couldn’t get from the simple one?

P: Oh… since I don’t know much about music, I don’t really understand it anyway. But, I think it would be pretty quick once you start clicking on a few bits to start figuring it out.

Declan: Awesome, thank you so much for taking the time to do this.

Interview 2 Transcript - Participant S

Declan: What previous experience (if any) do you have with digital music creation?

S: Zero.

Declan: What previous experience (if any) do you have with real-world music creation?

S: Zero.

Declan: What was your impression of the app upon first opening it?

S: It's nice and clear, I liked the erm… the layout. Because, for someone as myself, I need clear instructions. And I also liked the fact that… based on the person I am that I might rush ahead without having had full knowledge of reading everything that I should have… I liked that I had two ways to be able to check all those things out.

Declan: How intimidating was the application's functionality to find, understand, and use?

S: *sigh*, *laugh*, everything was there and explained, perfectly well. Once again, I would have to say that what worked really for someone as myself, was having the ability to perhaps do something I shouldn't have done, and then it could be corrected without anything going wrong. If that makes sense?

Declan: So it was reassuring that you could not *break* break it?

S: Absolutely, 100% correct!

Declan: What (if anything) frustrated you about using the app, or didn't work as you were expecting it to?

S: I think the only thing about that was more an interpretation problem for myself, was it took a while to grasp that, when I stopped it, that that didn't mean it automatically returned to 0. And then also, having further explanation with a partner that could use it, he was able to explain to me why obviously that's a good thing and not a bad thing. It's a positive thing not a negative thing.

Declan: But, for you personally it wasn't what you were expecting?

S: I think I just, yes… it just was… yes. I suppose, yeah.

Declan: What (if anything) did you enjoy about using the app?

S: I like the fact that I could create something that I'd never done previously before. And that I had an opportunity to keep altering that. To… actually generate sound.

Declan: That's very nice, I'm glad! Did you spend more time using the more detailed readout, and if so, did you learn anything from it?

S: I think again for me, that would have been secondary. And I think that then showed, how much I had enjoyed being a non-music person. The enjoyment of being able to create a sound, then created a secondary interest in me, to look and see what those sounds meant, when I changed various buttons. So, absolutely, I looked at terminology…

Declan: …so did you end up looking at what any of that meant, were you able to start piecing together 'oh this sounds like this, so that means that'?

S: I think again, coming from a non-creative-music person, for me it more was… made an interest for me to see, I don't know, maybe a glimpse into a world I don't know anything about. To see terminology, of how, by putting different bits together; different notes together and then adding different sounds, and tempos, and speeds. So those words then made a bit of sense, when I'm using the application, as in, rather than being, let's say Japanese or something.

Declan: They weren't totally abstract, you could link them to what you were hearing?

S: Correct!

Declan: Excellent, Thank you

Interview 3 Transcript - Participant K

Declan: What previous experience (if any) do you have with digital music creation?

K: None.

Declan: What previous experience (if any) do you have with real-world music creation?

K: Not too much, I've only tinkered about on a piano occasionally, and I'm learning guitar, so I guess that counts!?

Declan: What was your impression of the app upon first opening it?

K: I liked it. I liked the colours… I liked the colour scheme. And I liked that it was very straightforward looking. There wasn't anything that… it was just a nice, simple interface. Which I like.

Declan: How intimidating was the application's functionality to find, understand, and use?

K: Not intimidating at all. It was very easy to understand and use.

Declan: What (if anything) frustrated you about using the app, or didn't work as you were expecting it to?

K: I don't think anything frustrated me about the app. I think that I was expecting it to be… quicker. Like, the change of the sounds to be quicker. But I think that that's just, my personal preference.

Declan: So, more immediately obvious what's changed?

K: Yeah, like when I pressed the add note button. My expectation was that I would hear the change immediately, but it didn't happen. But I think that's just the nature of generative music, and I'm not used to that because I still don't fully understand what generative music is. So, I just think that my expectation of it was not, what it actually is. If that makes sense?

Declan: It does! What (if anything) did you enjoy about using the app?

K: I liked the sounds… that it produced. And, I liked that I could see - because I only have a little bit of knowledge about music - it was actually nice to see what it was actually doing behind the scenes. Yeah.

Declan: If using the app more, how much time do you think you would spend on the more detailed readout, and, did you learn anything from it when you did use it?

K: I would… I think I would spend more time on the detailed, going forward. I think if I spent more time on the app, I think I would spend more time looking at the detailed thing because I would want to know what was actually happening.

Interview 4 Transcript - Participant M

Declan: What previous experience (if any) do you have with digital music creation?

M: None.

Declan: What previous experience (if any) do you have with real-world music creation?

M: None, other than watching other people make music.

Declan: What was your impression of the app upon first opening it?

M: Very impressive!

Declan: How intimidating was the application's functionality to find, understand, and use?

M: It wasn't intimidating at all, I found it quite easy to add and subtract and to go up and down. To follow the instruction that it gave on the top when you clicked on the box. Not intimidating at all.

Declan: What (if anything) frustrated you about using the app, or didn't work as you were expecting it to?

M: The old laptop I was using it on! Nothing frustrating about the app itself.

Declan: What (if anything) did you enjoy about using the app?

M: I enjoyed listening to it. I liked the way it works!

Declan: If using the app more, how much time do you think you would spend on the more detailed readout, and, did you learn anything from it when you did use it?

M: Just, how it functions… basically. Well, not how *it* functions, but how I could make it function up and down…

Declan: What do you mean?

M: Well, by reading the script on the top when you click on a box it tells you what's going to happen. After that, a few times, you know what each box… what each box is going to do what.

Declan: So do you think you would use the more detailed version going forward?

M: A more detailed version?

Declan: Yes, the one which lists the notes, the key, etc…

M: I suppose, yeah… I'd use it some. But I'd use the one with the plus and minus, the more simple layout. I don't need to overcomplicate it because I know how it works now.

Interview 5 Transcript - Participant E

Declan: What previous experience (if any) do you have with digital music creation?

E: Right so, does BandLab count?

Declan: Probably, I'm not sure what it is?

E: So, the only experience I've had with creating digital music… part of my son's homework over lockdown, for music, we had to use this kind of app-thing. It was a website, and it was called BandLab. All you do is, you go on, and you just make music by clicking certain… you can choose instruments, and then you choose notes, and then you just click on it and it comes up like a dashboard, almost like a synthesiser on screen. So, we've played around with it, which is the only experience I've had.

Declan: Yeah that would definitely count!

E: It was really cool, it was good fun!

Declan: What previous experience (if any) do you have with real-world music creation?

E: Apart from plinking on my daughter's piano… I used to play the recorder, I used to play clarinet, I had piano lessons for a little while. We've got a piano in the house, and I will from time to time go and play on that. I won't necessarily - because I can't read music, well, I suppose I can read a little bit of music - but I normally just play around on it myself. I just, plink around on it, my own plink-plonk music! That's what I do. I like it, it's relaxing *laugh*!

Declan: What was your impression of the app upon first opening it?

E: So when I opened it, and I saw the… so you've got the symbols… Yes, when I opened it, it was the symbols on each of the buttons. I got the green play, and the red stop straight away. And then I thought, "oh gosh, I wonder what these other buttons do?" So, I just started clicking on them *laugh*! And then pressed play, and then thought "wow, it makes a sound!" So, I thought it was really… the way it looks, I like it, it's really simple, it's really neat. And then, of course you can just click on the black, kind of, space at the top where it's got the text. And then it changes it, and then you know exactly what these buttons do. But, I'm more of a just click the buttons and see what happens sort of person. Then on the proviso nothing bad happens, then I'll try and have a look to see if there are any instructions.

Declan: So, to follow up with that, did you feel confident nothing bad was going to happen?

E: Yeah, absolutely. Yeah. Because the words at the top, were quite… I liked it, it told me, "try all the buttons". So I did! "Try all the buttons, go slow, it's chill!" I thought cool, let's have a play.

Declan: How intimidating was the application's functionality to find, understand, and use?

E: Not at all. The one that puzzled me most was the change sound button. When I first looked at it I didn't know what that was at all. So, plus and minus were fairly, y'know, having read the text at the top, plus and minus were fine. Then, up and down I thought might be volume at first, and then the kind of 'z' on a slant, I didn't know what that was at all, but I wasn't intimidated, at all. No, it was fine.

Declan: What (if anything) frustrated you about using the app, or didn't work as you were expecting it to?

E: When I first clicked… I clicked quite a few buttons at the start, like adding notes and changing the sound and the tempo, without knowing what I was doing, because I didn't have the text on the buttons at first. Then I played it, which was fine. But then I pressed stop, but it didn't stop! But then it said… I think it was on the text at the top… but it did say it might, I can't remember the exact words?

Declan: The message about the extra sounds fading out?

E: That's it. Yeah. But I didn't know that until I looked for it. So when I pressed the stop button and it didn't stop, I wasn't irritated but I was like "oh my god!" I thought I'd broken it!

Declan: Right, it was a bit disconcerting?

E: Yeah, but then when I read that bit I thought "ok then it's fine, that's fine!"

Declan: What (if anything) did you enjoy about using the app?

E: The simplicity of it. I thought it was really good. I mean it's easy, and it's quick! So, I am the sort of person that likes quick output for very little effort. And this was really cool, so you could really just click a few buttons and you can play music. And then you can start again, and you can take notes away, and change it very easily. I liked it!

Declan: If using the app more, how much time do you think you would spend on the more detailed readout, and, did you learn anything from it when you did use it?

E: So, the detailed readout that tells me the different types of note sounds and keys? Yeah… I would spend some time. Because I think you could generate some quite nice music from this actually, some quite nice sounds. So I would, I would spend time looking at this more detailed stuff because I'd go through the different sounds, to understand them. And then, I would feel like I could, more… not use it intuitively, but in a more planned way, if that makes sense? So I could think more, and be more methodical in my approach to which buttons I might click. Rather than just generating any random sound, I guess I could be less random by using the information that it's giving me about changing the tempo, and changing the key, and changing the actual sound itself. You could actually try to compose something rather than just randomly generate something, if that makes sense?

Declan: It does. So following up on that, would you ever take what you've seen or heard, outside of the app? Or do you think you'd be able to, does it feel more contained in the experience of the app?

E: So, I could probably learn… when I'm plink-plonking on my piano… I could probably, I might learn something from this app in terms of key, sound, if that makes sense, and tempo? I could maybe take some of that learning from using the more detailed text and I could probably take some of that learning when I'm playing my piano.

Declan: Hopefully, that would be good!

E: Well, I quite like it, it is a neat app *laugh*!

Interview 6 Transcript - Participant N

Declan: What previous experience (if any) do you have with digital music creation?

N: Quite a bit! I use DAWs regularly, I have made my own app that makes digital music. So I would say quite experienced!

Declan: What previous experience (if any) do you have with real-world music creation?

N: I've been writing music for - don't even know how long - two digits, I'm sure, years. So, plenty.

Declan: What was your impression of the app upon first opening it?

N: It's very obvious, what's going on. Very simple to use. Big buttons! Which is good, it's very accessible.

Declan: How intimidating was the application's functionality to find, understand, and use?

N: Not intimidating at all. The, advanced mode, was a bit more complex, not really intimidating. I can see why someone else might find it… if they somehow ended up in advanced mode…

Declan: Right, but not for you?

N: Not for me in particular, no. I think it was pretty straightforward.

Declan: What (if anything) frustrated you about using the app, or didn't work as you were expecting it to?

N: Umm… nothing in particular. It all worked as expected, it all seemed straightforward!

Declan: What (if anything) did you enjoy about using the app?

N: How quickly you can set something up to just make some cool sounds. You just do two-three clicks, and you've got something going on.

Declan: Is that, would you say, in contrast to other apps, or just in this app itself?
N: Just with the app itself, you can open it and in two seconds you've got something.

Declan: If using the app more, how much time do you think you would spend on the more detailed readout, and, did you learn anything from it when you did use it?

N: I would, I'd spend most of my time in the advanced mode. I did learn quite a bit of how it worked.

Declan: I'm guessing with your previous experience, it wasn't the music theory it was teaching you, it was more about how the app itself was working?

N: Yeah, more about how the code was. I did have a look at the code after - but not having had a look before opening it, it was, illuminating, I guess. You could see quite a bit of what was going on, which was very interesting. In terms of music theory, yeah, not for me in particular.

Declan: Yeah, it's not very advanced stuff…

N: Yeah, but you would learn something if you didn't know a whole lot about it previously.

Declan: Hopefully yeah, it was trying to pitch a less scary amount of complexity, if you didn't know anything going in. But having as much experience as you do, there's probably not much this can show you that's new!

N: In terms of both, the technology and the music, in combination you can always pick something up, which is what I felt about it.