Abstract

The thesis encapsulate the design and implementation of an interactive software synthesizer, built with the intent to serve as an accessible tool for exploring an alternative music tuning system, Nineteen-tone Equal Temperament (19-TET) . Fairly large portions of the text aim to summarize and explain certain principles of music theory, in order to provide readers unfamiliar with the subject a foundation of knowledge. Although musical tuning systems can be compared and discussed purely theoretically, which is also done in this thesis, actually hearing the differences was considered more interesting. This is the reason for which the instrument was developed. Although the work does not include any comparative study of the tunings as such, the instrument developed is intended to be of help during such studies. The larger challenge within the course of this project has been designing the instrument's user interface. This could be considered the main goal of the project.

Referat

En synthesizer för nittontonsskalan

Examensarbetet sammanfattar design och implementation av en interaktiv mjukvarusynth, byggd med avsikten att fungera som ett lättbegripligt verktyg för att underlätta utforskandet av en alternativ stämning, nittontonig liksvävande temperatur. Förhållandevis stora delar av texten ämnar summera och förklara vissa musikteoretiska principer, för att förse ovana läsare med en kunskapsgrund. Även om stämningar kan jämföras och diskuteras på ett helt teoretiskt plan, något som också görs i detta arbete, vore det mer intressant att faktiskt höra skillnaderna. Detta är skälet till att detta instrument utvecklades. Även om arbetet inte innehåller någon jämförande studie med avseende på stämningarna i sig, så är instrumentet som utvecklats tänkt att kunna vara till hjälp under sådana studier. Den större utmaningen under projektets gång har varit att ta fram instrumentets användargränssnitt. Detta kan anses vara projektets huvudsyfte.

Contents

Theoretical background

Much of the work during the course of this project has consisted of researching fundamental principles of music theory. Concepts such as tuning system, intervals, equal temperament and more had to be understood, simply in order to have a chance at succeeding with the project. This section aims to reiterate the main points of these concepts in order to familiarize the reader with them, as they are frequently used and referred to throughout the text. Brief explanations of most terms used in this section should be available in the glossary.

1.1 Tuning and equal temperament

There is a need to explain what exactly is meant by "nineteen tone scale". Generally speaking, music is based on some tuning system. Basically, a tuning system might be a set of rules or maybe just a mathematical formula, defining what pitches a correctly tuned instrument should produce. In one particular tuning system, called Twelve-tone Equal Temperament (12-TET), adjacent notes of the chromatic scale are all separated by logarithmically equal distances, meaning that the frequency distance between adjacent pitches increases upwards the scale, but the ratio between them is always the same, $2^{1/12}$ or 100 *cents*. An interval, the ratio between two pitches is usually expressed in cents is defined as

$$n = 1200 \log_2 \left(\frac{a}{b} \right) \tag{1.1}$$

where $\frac{a}{b}$ is the ratio between them. Thus the interval between two adjacent tones in 12-TET, commonly called a semitone, is always $n = 1200 \log_2 \left(2^{1/12} \right) = 100$.

12-TET is the most commonly used tuning system used in Western music, and is the standard system for tuning a piano. The note names of the twelve pitch classes are: $C, C\sharp/D\flat, D, D\sharp/E\flat, E, F, F\sharp/G\flat, G, G\sharp/A\flat, A, A\sharp/B\flat, B$. The reason for using these names will be disclosed further on. The following definition is used to find the pitch or frequency of a note in 12-TET:

$$P_n = P_a \left(\sqrt[12]{2} \right)^{k-a} \tag{1.2}$$

where P_a refers to the frequency of a reference pitch[1]. The variables k and a refer to to numbers assigned to the notes. The pitch A4 is usually assigned the number 49, as A4 is the 49th key on a regular piano [3]. Counting backwards from A in the list of pitch classes above, it is found that C4 maps to key number 40. The frequency of the pitch C4 is thus

$$P_{40} = P_{49}\left(\sqrt[12]{2}\right)^{40-49} = \frac{440}{2^{9/12}} \approx 261.63 \text{ Hz} \tag{1.3}$$

Nineteen-tone equal temperament, **19-TET**, is not very different from 12-TET. It has 19 pitch classes instead of twelve:
$C, C\sharp, D\flat, D, D\sharp, E\flat, E, E\sharp/F\flat, F, F\sharp, G\flat, G, G\sharp, A\flat, A, A\sharp, B\flat, B, B\sharp/C\flat$. Instead of having the five enharmonic pairs $C\sharp/D\flat$, $D\sharp/E\flat$, $F\sharp/G\flat$, $G\sharp/A\flat$ and $A\sharp/B\flat$, these are now considered separate. In addition $E\sharp/F\flat$ and $B\sharp/C\flat$ are introduced, yeilding a set of 19 pitch classes instead of 12. Nineteen-tone Equal Temperament (19-TET) is mathematically very similar to 12-TET, the difference is that in 19-TET, the smallest interval ratio is equal to $\sqrt[19]{2}$ instead of $\sqrt[12]{2}$.

1.2 Overtones and harmonic series

To understand what makes combinations of tones to be perceived as dissonant or consonant, there is need to understand a few things about harmonic sound. Sound can, at least theoretically, be modeled as sums of different sine waves of different frequency, amplitude and phase. A sawtooth-shaped wave can theoretically be constructed summing a large number of sine waves in a manner most generally expressed as a infinite harmonic series [5]:

$$x_{sawtooth}(t) = \frac{\pi}{2}\sum_{k=1}^{\infty}\frac{sin(2\pi kft)}{k} \tag{1.4}$$

The partials of this series are the odd and even harmonics or overtones of the same fundamental frequency f. Informally, harmonic sound is sound that is perceived as one unified tone as a result of the waves being periodical, in contrast to disharmonic sound, such as noise, in which the wave is completely random. Harmonic sound can be created by adding sine waves of the same harmonic series and it can model the behaviour of physical instruments to some extent. As one increase the number of overtones, the sound will be perceived as "sharper". What causes the perceiving of two or more tones as being dissonant or consonant is decided by how their overtones coincide. For example, consider two different sawtooth waves with fundamental frequencies $f_0 = 200$ Hz and $f_0' = 300$ Hz, then they will also have overtones $f_1 = 400, f_2 = 600, ...$ and $f_1' = 600, f_2' = 900, ...$ But f_2 coincides with f_1', they are both 600 Hz. Hearing these waves will be perceived as very consonant, and the reason is that they have coinciding overtones [15]

[1]The reference pitch is usually A4, which normally has the frequency 440 Hz [1].

1.3 Intervals, the diatonic scale and just intonation

To understand in what ways different tuning systems differ from one another, and how to compare them to each other, it is helpful to understand the concept of *just intonation*. In music, just intonation is any musical tuning system in which the frequencies of notes are related by *different* ratios of small whole numbers, in contrast to equal temperament, in which an interval is defined as a multiple of the same basic interval. Any interval tuned this way is called a *just interval* [7]. The two notes in any just interval are members of the same harmonic series, like in the example of previous section. In fact, the ratio between the two tones in the example is the just interval 3/2, also known as a *perfect fifth*. Three basic intervals can be used to construct any interval involving the prime numbers 2, 3 and 5 (known as the *5-limit intonation*), defined in table 1.1.

Name	Ratio
Semitone	$s = 16/15$
Minor tone	$t = 10/9$
Major tone	$T = 9/8$

Table 1.1. The 5-limit intonation

These are multiplied in various ways to form several other just intervals, see table 1.2. A just diatonic scale can be derived from these intervals, see table 1.3.

Name	Ratio
Minor third	$6/5 = Ts$
Major third	$5/4 = Tt$
Perfect fourth	$4/3 = Tts$
Perfect fifth	$3/2 = Ttts$
Octave	$2/1 = TTTttss$

Table 1.2. Intervals derived from the 5-limit intonation

Note	Ratio
C	$1/1$
D	$9/8 = T$
E	$5/4 = Tt$
F	$4/3 = Tts$
G	$3/2 = TTts$
A	$5/3 = (Tt)(Tts)$
B	$15/8 = (Tt)(TTts)$
C	$2/1 = (Tts)(TTts)$

Table 1.3. A just diatonic scale

Many musical tuning systems, including 12-TET and 19-TET, are based on the diatonic scale. On a piano keyboard the diatonic scale is mapped directly onto the white keys, C, D, E, F, G, A, B. The intervals between the tones of the diatonic scale is T-T-s-T-T-T-s, using the notation in table 1.1.

It can also be written C-T-D-T-E-s-F-T-G-T-A-T-B-s-C. In order to acquire the full chromatic scale, five additional keys are needed, and naturally, they are placed *between* the keys that are a major tone apart. Since E and F resp. B and C are only a semitone apart, there should not be any black key between them. The pitches of these five black keys are considered to be either *flat* or *sharp*. For example the key between C and D is either a sharp C, $C\sharp$ or a flat D, $D\flat$.

In order to understand the *purity* of a particular tuning system, it is usual to examine how well intervals in that tuning system approximate just intervals. What could generally be strived for developing a new tuning system is getting as many well approximating intervals as possible. The reason for not simply using just intonation instead equal temperament is that it can produce some rather awkward interval ratios when combining intervals, it also does not support the transposing of intervals, making it very difficult and impractical [7]. A brief examination of both 12-TET and 19-TET is contained within the next section.

Introduction

Now that important principles and concepts have been breifly explained, it should be appropriate to finally present the actual project, what it is about and why it is carried out. To begin with, a purely mathematical account for the differences between 12-TET and 19-TET does not really provide the whole picture. A comprehensive study of these tuning systems and their differences should probably involve some audible examples. For example, an audio adaptation of the interval tables in the previous section would positively help widen the view, but it is felt that this would not present enough challenge, it would not be very interesting either. To appreciate the difference between 12-TET and 19-TET, an instrument capable of demonstrating both would be of great use, and since there is not really any such instrument available, it would have to be built, and in this case, purely in software. The remaining part of this text will deal with the development of this instrument, and present the main problems encountered during the process. The actual synthesizer (here referring to the component that produces sound), will be implemented using the ChucK audio programming language. ChucK comes with a standard library containing some very well-implemented high-level components for developing audio synthesis. This is good because dealing with signal processing and real-time audio synthesis in particular using a general purpose programming language is hard and complicated. Moreover, not having to put in the time and effort to implement a real-time audio synthesis from scratch leaves more time to design and develop the user interface of the instrument.

2.1 Differences between 12-TET and 19-TET

Since very early, composers and theorists have been experimenting with different tuning systems, but more recently, 19-TET frequently emerges as one of the more plausible alternatives to the widely used 12-TET [15]. As briefly discussed earlier, 19-TET is the tempered scale in which the octave is divided into 19 equal steps instead of 12. Basically, in 19-TET, each step represents a frequency increase by a factor $\sqrt[19]{2}$ instead of $\sqrt[12]{2}$ as in 12-TET. Composer Joel Mandelbaum examines the acoustical properties of the 19-TET tuning, and advocates for its use [14]. Mandelbaum demonstrated why he believed this system to be the only reasonably system

with a number of divisions between 12 and 22 [12]. Mandelbaum explains why 19-TET in particular should be considered as an alternative for 12-TET. Here focusing on the acoustical aspect, the most apparent changes observable when switching from 12-TET to 19-TET is that the fifth and fourth intervals are somewhat impure, in return for the third and sixth interval which are much closer to just thirds and sixths than their relatively dissonant counterpart in 12-TET. Basically, both tunings have some flaws and some strengths [14]. Presented below are some intervals in 12-TET and some in 19-TET both approximating some common just intervals. The errors are simply the differences between the actual 12-TET or 19-TET intervals and the just ratios in cents, see 2.1

$$\Delta = 1200 \left(\log_2 2^{s/N} - \log_2 \frac{a}{b} \right) \tag{2.1}$$

where N is here either 12 or 19, s is the number of steps and $\frac{a}{b}$ is the just ratio.

Name	Steps	Ratio	Just ratio	Size (cents)	Just size (cents)	Error
Minor second	1	$2^{1/12}$	16/15	100	111.73	-11.73
Major second	2	$2^{2/12}$	9/8	200	203.91	-3.91
Minor third	3	$2^{3/12}$	6/5	300	315.64	-15.64
Major third	4	$2^{4/12}$	5/4	400	386.31	+13.69
Perfect fourth	5	$2^{5/12}$	4/3	500	498.04	+1.96
Perfect fifth	7	$2^{7/12}$	3/2	700	701.96	-1.96
Minor sixth	8	$2^{8/12}$	8/5	800	813.69	-13.69
Major sixth	9	$2^{9/12}$	5/3	900	884.36	+15.64

Table 2.1. Common intervals in 12-TET, compared to just ratios

Name	Steps	Ratio	Just ratio	Size (cents)	Just size (cents)	Error
Diatonic semitone	2	$2^{2/19}$	16/15	126.32	111.73	+14.59
Major tone	3	$2^{3/19}$	9/8	189.47	203.91	-14.44
Minor third	5	$2^{5/19}$	6/5	315.79	315.64	+0.15
Major third	6	$2^{6/19}$	5/4	378.95	386.31	-7.36
Perfect fourth	8	$2^{8/19}$	4/3	505.26	498.04	+7.22
Perfect fifth	11	$2^{11/19}$	3/2	694.74	701.96	-7.22
Minor sixth	13	$2^{13/19}$	8/5	821.05	813.69	+7.37
Major sixth	14	$2^{14/19}$	5/3	884.21	884.36	-0.15

Table 2.2. Common intervals in 19-TET, compared to just ratios

It is clear that 19-TET indeed gains accuracy in the thirds and the sixths. On the other hand it loses accuracy in the fourth, fifth and major tone intervals. But really, what do these differences mean; how bad is it really that the fifths of 19-TET is around seven cents off? Questions of this kind are discussed in the next section.

2.2 The ChucK Audio Programming Language

Since ChucK is integral to the implementation, and since the language is very new and unusual in some aspects, this section provides a short analysis of the most important principles and concepts of the language. Although reading this it is not necessary in order to understand the main points of the work, it might still interest some readers.

Events are one of the central programming concepts in ChucK. In ChucK, there are *shreds*, which are a kind of process, running in the ChucK virtual machine. Events provides a means to communicate between different shreds. Listing 2.1 briefly illustrates some of the most central and unique aspect of the ChucK language. Two event objects are created, and two function handling these events are defined. The expression `e1 => now;` is blocking and causes the executing shred to suspend (and let other shreds run on the VM) and be placed at the back of the event's waiting list. Eventually, in this example, the main shred will signal the event and the handler will print something.

The expression found on lines 26 and 29 is very important. It means that *time will advance* by 500 milliseconds. This in turn means that the shred will suspend for that exact amount of time. It also means that other shreds are allowed to compute audio for that exact amount of time.

Basically, to get ChucK to produce sound, *time must advance*, and this is the way it is done. On line 21 a sine oscillator has been connected to a gain function that is connected to the `dac`[1]. Running this program will result in repeated beeping and printing. This is not very exciting, but in fact, this example illustrates most concepts needed to understand all ChucK code contained within this document, given of course that the reader already has some experince with reading code written in not so familiar programming languages.

Listing 2.1. Example illustrating concurrency, events and audio programming in ChucK

```
1  Event e1, e2;
2
3  fun void e1Handler() {
4    while (true) {
5      e1 => now;
6      <<< "handled␣event␣e1" >>>;
7    }
8  }
9
10 fun void e2Handler() {
11   while (true) {
12     e2 => now;
13     <<< "handled␣event␣e2" >>>;
14   }
15 }
```

[1] In general, dac stands for digital/analog converter. In the context of ChucK programming, connecting some unit generator to `dac` will enable actual sound, as soon as some audio is computed.

```
16
17 spork ~ e1Handler();
18 spork ~ e2Handler();
19
20 0 => int c;
21 SinOsc s => Gain g => dac;
22 s.freq = 440;
23
24 while (true) {
25   0.4 => g.gain;
26   500::ms => now;
27
28   0 => g.gain;
29   500::ms => now;
30
31   if (c++ % 2) {
32     e1.signal();
33   } else {
34     e2.signal();
35   }
36 }
```

Problems

There are two main problem categories recognized, each separately discussed.

3.1 Comparability of tuning systems

There is a problem that concerns the perceiving of 19-TET in contrast to 12-TET. As seen, the main differences can be presented as very exact data, derived using only the mathematical definitions of the tuning systems. However, as previously touched on, the data obtained in tables 2.1 and 2.1 above might not provide the whole picture. The data is abstract and purely quantitative, it can be difficult to realize how these differences manifest themselves, qualitatively speaking. For example, how *much* worse does the perfect fifth in 19-TET *sound* compared to the perfect fifth in 12-TET? Generally, how are the differences perceived and which tuning is perceived as most consonant? Studies set out to answer questions of this kind has been done, but results were somewhat inconclusive [15]. Consequently, the purpose of the thesis is not to determine what tuning is the "better" one, but rather to work out the design of an instrument that supports both tunings, and that has the abilitity to switch between them. In order to be absolutely clear on this point, there will be no attempts to dismiss one tuning in favour of another. So, the questions regarding the respective qualities of the two tunings will remain open, and should be investiged further, but that would have to be in some future study. However, the instrument developed in this project could definitely serve as a practical tool in such a study.

3.2 User interface

Since the maths needed to compute the note-to-frequency mappings can be considered fairly trivial, and everything concerning real-time audio synthesis is easily realized using ChucK, emphasis has been applied to questions concerning user interface. Since it will be software based, the possibilities are restricted. Not everything can be done, and not everything is practical. Immediately, questions concerning playability emerges – How should this instrument be played? Is there any previous similar work that can help and inspire? And since it is software based, which

of the computer's input device(s) should be used? Secondly, what is a suitable graphical user interface for this instrument and purely practical; How should it be implemented? What development tools should be used? ChucK will stand for the synthesis part but ChucK does not really have support for Graphical User Interface (GUI) development, how is this solved?

Method

The aim of this section is to provide answers to the questions asked in the chapter Problems. It is divided into subsections, each treating their own category of problems. There is not really an order of priority here, but the ordering of these subsections aims to pass through as natural and logical.

4.1 Input devices

It is obvious that this instrument needs some kind of input in order to provide flexible functionality. Implementations could be considered in which some kind of notational data is read from files. While such solutions should be easy to implement, they would suffer from complete lack of interactivity. The process of programming such an instrument would most likely be tiresome and counterintuitive. There is a need for some kind of interactive input.

So what devices can be of use? Thoughts of using a regular MIDI-keyboard got ruled out quickly due to the fact that they are not really suited for any scales other than those having twelve tones. Some inconvenient remapping of the keys would in the end only confuse and obstruct any sensible usage. This basically leaves the mouse and the computer keyboard, because at this point there is neither time nor knowledge to create any custom input devices, although this would really be the ultimate. Using only the mouse to interact with a GUI will surely work, but it would lack playability. Striking a chord would become virtually impossible. So what about the computer keyboard? It has a lot of buttons so at least one octave of 19-TET scale could easily be mapped to some section of it. ChucK has excellent support for keyboard input as well, it catches system-wide keyboard events, meaning the ChucK application will not have to be in focus while using it. It is known that the computer keyboard will fail to support truly polyphonic playing; it cannot register state changes on more than three keys at once. Still, it remains the best alternative, because playing the keyboard merely by clicking is a lot worse.

4.2 Keyboard design

It is established that the computer keyboard is the only viable input device to be considered, but how should the keys on it map onto the pitch classes of 19-TET? It is fair to say that this problem has been the most time-consuming. Research eventually led to the notion of two possible keyboard layouts explicitly designed for instruments using a nineteen tone scale. One that really seems to make sense is the design found in the appendix of Joseph Yasser's book "A Theory of Evolving Tonality" [17]. In it, each of the black keys $C\sharp/D\flat, D\sharp/E\flat, F\sharp/G\flat, G\sharp/A\flat, A\sharp/B\flat$ of a regular piano keyboard is split into two separate keys. The synonyms, $(C\sharp/D\flat, D\sharp/E\flat, ...)$ is no longer, instead there is $C\sharp$ *and* $D\flat$, $D\sharp$ *and* $E\flat$ and so on. Two new black keys are introduced: $E\sharp/F\flat$ and $B\sharp/C\flat$, between E and F and B and the next C, yielding a total of 19 keys per octave. see figure 4.1 below.

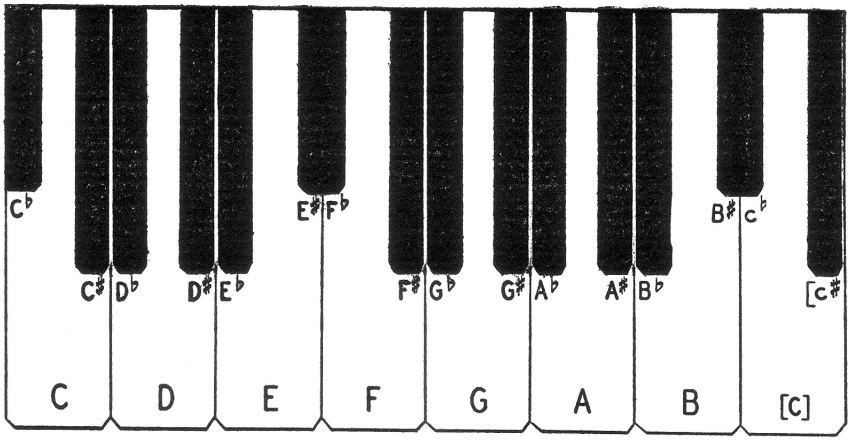

Figure 4.1. Yasser's 19-tone keyboard

While this design at first seemed aesthetically appealing and logical, it was soon found to be impractical in the context of this project. There is simply no natural way of transferring the layout of this keyboard onto a computer keyboard. If there were to be a GUI only, it would have been fine, but since real keyboard input has been established necessary, this design will not do. However, there is one other alternative. In 1975, Erv Wilson and Scott Hackleman collaborated in the construction of a clavichord using one of Wilson's generalized keyboard designs, carried out to 19 tones per octave [13] [16]. Generalized keyboards are musical keyboards with regular, tile-like arrangements usually with rectangular or hexagonal keys. The concept of the generalized keyboard was first proposed by Robert Bosanquet in the 1870s, and since the 1960s, Erv Wilson has explored it further, developing new methods of using and expanding them [4].

The theory of the generalized keyboard is not entirely trivial, but let us introduce it using an example. A generalized keyboard for the 19-TET scale could be

calculated in the following manner. Let G be the abelian group $(\mathbb{Z}_{19}, +)$ and let 7 be a generator:

$$G = (\mathbb{Z}_{19}, +) = \langle 7 \rangle = \{0, 7, -5, 2, 9, -3, 4, -8, -1, 6, -6, 1, 8, -4, 3, 10, -2, 5, -7\} \tag{4.1}$$

In unchanged order, these numbers determine the vertical positions of the keys. Flat pitches are assigned the largest negative member of its congruence class[1] rather than the smallest positive number, causing them to get positions below the others, see figure 4.2.

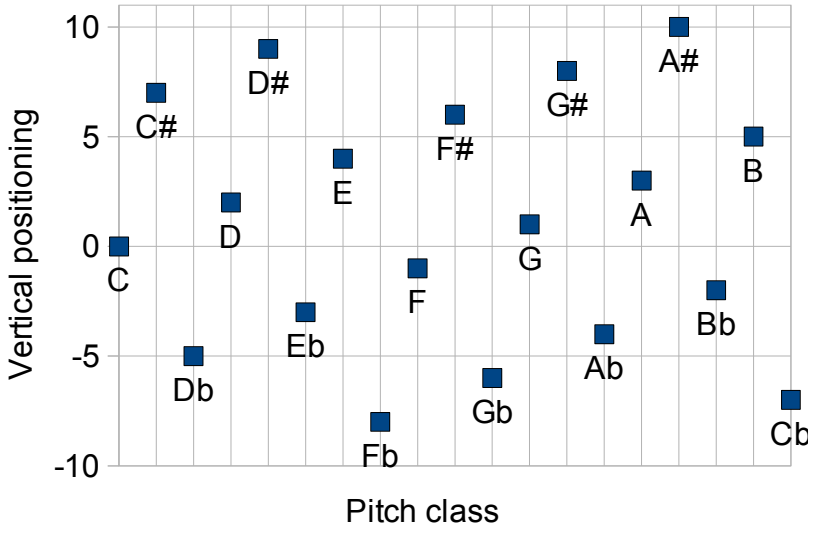

Figure 4.2. Schematic of one octave of a generalized keyboard for 19-TET using a 7 as generator

How does this rather intricate mapping fit with the computer keyboard layout? Initially and completely unchanged, it seems not to. But if the vertical positions allows some displacement and realignment, an adaptation can be formed. Let the row of keys `A, S, D, F, G, H, J, K` map to the diatonic scale, $C, D, E, F, G, A, B, (C)$. Then let the row below, `Z, X, C, V, B, N, M` map to the flats $D\flat, E\flat, F\flat, G\flat, A\flat, B\flat, C\flat$, and the keys `W, E, T, Y, U` map to the sharps $C\sharp, D\sharp, F\sharp, G\sharp, A\sharp$. While this mapping does not fully conform to the generalized keyboard, it is an approximation, justifiable by practical reasons. Since the computer keyboard is really the only reasonable input device for this instrument, this seems like the only practical solution. See figure 4.3.

[1]The set of integers congruent to a modulo n. For example $7 + 7 \equiv_{19} 14 \equiv_{19} -5$.

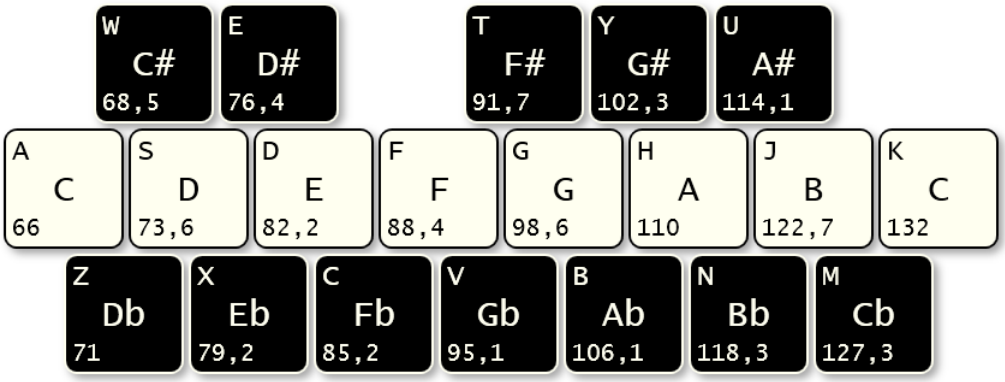

Figure 4.3. User interface keyboard adaptation of the generalized keyboard for 19-TET

In order to acquire a suitable keyboard mapping for 12-TET, simply remove the flat keys from the above mapping, and rename and remap the previously explictly sharp keys, see figure 4.4.

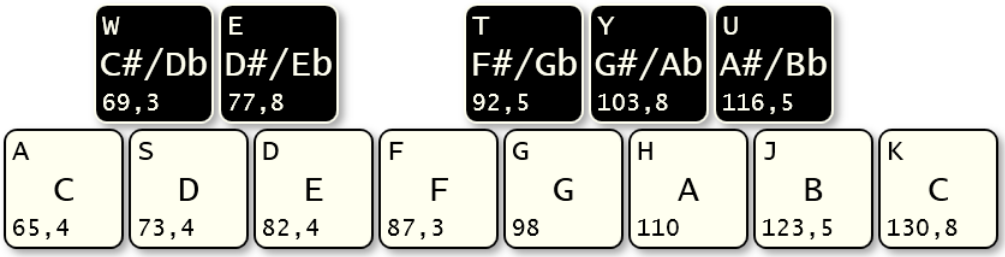

Figure 4.4. user interface keyboard adaptation for 12-TET

4.3 Implementation

As previously established, the actual synthesizer or the *back-end* component of the application is implemented using the ChucK audio programming language. As the main goals of this project does not include providing an in-depth analysis of the actual programming language, this section will provide a summary of the how and why the functional features were implemented. The ChucK language should be fairly easily understood by readers accustomed to programming languages and the chapter Introduction contains a short review of the most important aspects of the ChucK language.

4.3.1 The back-end

This subsection serves to describe solutions pertaining the back-end component, i.e the synthesizer. Sections below discusses the most important logical components.

Mapping keys to frequencies

Synthesis aspects of the application set aside for a while, the most important functionality is the mapping between the keyboard keys and the frequencies attached to them. This functionality is implemented in a class simply called `Mapper`. Handling keyboard input could have been done in the GUI component and then fed into the back-end component, but since ChucK has excellent support for handling this kind of input, it was decided to do it the other way around. The mapper however does not actually handle the keyboard input, but it is responsible for translating the input into frequencies later sent to the synth.

The `Mapper` provides one function, `remap(int numDiv)`. It can be called without arguments, or with the arguments 12 or 19. Calling it will cause a few things to happen. Firstly `setupKeys()` will be called. Selected frequency spans in the 12-TET and 19-TET scales are pre-calculated at the moment of instantiation of a mapper object, so mapping is really only a matter of copying certain frequencies into elements of the map indexed by the correct values, namely the ASCII code for the keys that are used.

Frequencies are computed applying a generalization of definition 1.2. Calling `remap` has one more effect. After `setupKeys()` has finished, the new mapping will be sent via OSC to the GUI component. This is because the GUI is designed to reflect, in detail, what scale is used at the moment and what the exact frequencies of the pitches are. The mapper is then used in the application by simply accessing the array, indexing it with ASCII values. Please refer to listing A.4 for details.

Synthesizers

To provide some level of modularity, in order to ease further development, some basic object oriented programming patterns were applied. Wishes to be able to rapidly implement more than one synthesizer within the program, without having to rewrite tons of code, led to the design of an *abstract* synthesizer class called `AbstractPolySynth`. Basically it is what the name suggest, an abstract class partly defining and partly declaring the functionality of a polyphonic synthesizer. It declares a set of functions that has to be implemented by any class that extends it. Examples of such functions are: `init()` which should implement all initializing operations such as sporking shreds and assigning member variables. Constructors are not yet implemented in ChucK, but basically what would have been done in a constructor is done in the `init()`-function. The `voiceHandler()` is a function that will run in multiple separate shreds, each responsible for one of the synthesizers voices.

Moreover, there are some function common to all synths, and therefore defined in this abstract class. Examples of such functions are the `stop()` function, which will kill all shreds belonging to a synth, and `disconnect()` which will disconnect the synthesizer from whatever output it was connected to during its initialization. There are two very important common functions defined in the abstract synth class called `noteOnHandler()` and `noteOffHandler()`. These are also running in separate shreds, and they listen for OSC messages sent by the GUI component.

For example, whenever the OSC message `/note/on,i` is sent, meaning that the user pressed one of the GUI keys, the `noteOnHandler()` in turn will signal the `keyDown` event, which is a member variable of the abstract synth class. Implemented correctly, the `voiceHandler()` listens for the `keyDown` event and when one of the shreds running `voiceHandler()` catches it, it will cause the unit generator[2] for its assigned voice to start producing sound. Again, please refer to actual code found in listings A.5 and A.6 for more details on this.

Utility classes

There are a few couple of classes left. One is called `ShredCollection`. A synthesizer that needs to be able to be stopped needs to be able to remove its shreds from the virtual machine. Removing shreds is done by using the ChucK function `Machine.remove(int id)`, but one need to provide the id of the shred to be removed. By having each handler calling the `register(int id)`-method of the synth's `ShredCollection`-object, storing the id's, the shreds later can be removed from the ChucK VM by calling `ShredCollection.killall()`. Of course this solution is fraught with potential problems, since just forgetting to register a shred in a handler will result in "zombie" shreds whenever the synthesizer is killed, but it is what felt plausible at the time, it took only about 20 minutes to implement and it works fine if used correctly.

Another utility class is the `OSC` class, which basically just bundles together an instance of the `OscRecv` class, which is provided by the ChucK standard library and can be used to *listen* for and *receive* OSC messages, and an instance of the `OscSend` class, which can be used to *create* and *send* OSC messages.

Putting it all together

Now that the most important ChucK classes of the application has been described to some extent, it should be all right to take a look at the main part of the program. This part of the program is found in listing A.7. Initially, it instantiates and initializes objects of the classes recently discussed. Then it tries to open an interface to the computer's keyboard device. If this is successful, the program now gets keyboard events system-wide, meaning that ChucK will not have to be in focus, it can run somewhere in the background.

[2]The common name for classes found in ChucK's standard set of unit generators, classes that processes audio, such as the `SinOsc` in listing 2.1

Then the program enters a loop fetching keys events from the interface. The inner loop takes key events and performs the appropriate actions. For example if the `up`-key is pressed, the program will cause a remap of the keyboard so that all pitches are increased by one octave. If `enter` is pressed, the program will cause a remap from 12-TET to 19-TET or vice versa. If the key pressed is one of the keys currently mapped by the mapper, the `keyDown` event will be signaled, causing one of the voiceHandlers to wake up and do its work. This event will also cause the program to *send* an OSC message to the GUI, so it can graphically reflect that a key is pushed.

4.3.2 The graphical user interface

It should be said that the GUI initially was planned to be developed using the Python programming language with the help of some interesting Application Programming Interfaces (APIs) found. However, back then the development was also planned to be done under Linux, but ChucK turned out working a lot better running under Microsoft Windows®, which then quickly was decided to be the default working platform. The decision to use Windows as platform in turn led to the decision to use Microsoft Visual Studio 2010®, C#/.NET®and WPF®since there was good knowledge of this environment. Also, a very good free API for the OSC protocol was found [8].

What is actually seen on figures 4.3 and 4.4 is the actual keyboard part of the final GUI. The black and white keys are the visual part of a custom developed user interface control simply called KeyControl. See listing A.11 for code-behind and listing A.16 for XAML markup code.

It implements a set of different functions which will now be shortly described. First off, each `KeyControl` object has a few visual properties. It is either black or white, as it might represent sharp, flat or neutral pitch. It has a letter in the upper left corner, which denotes its physical equivalent on the real keyboard. In the lower left corner there is a number denoting the frequency of the pitch currently mapped to the key. Right on the key in the middle is the actual note name, or pitch class. If the user pushes left mouse button down while hovering a key control, the key will appear to actually be pushed down a bit, using some nifty animation possibilites that comes with the WPF framework.

More importantly, the GUI component will send an OSC message to the back-end, causing it to play the tone. In the same way, if the user presses the keyboard, switches octave or changes tuning (actions that are taken care out directly by the back-end), these changes will immediately reflect in the GUI, since the back-end sends OSC messages as well. The GUI, just like the back-end, is largely event-driven. There is a static class defining used OSC messages[3], and there is a Server class[4] that receives incoming OSC messages and invokes the appropriate actions.

[3]See listing A.8
[4]See listing A.9

The `Keyboard` control is simply arranging a set of `KeyControl` object in such a way that it maps correctly to the real keyboard.

In addition to this, the GUI contains a panel with sliders used to modify any paramaters of the synthesizer. These of course also need to make use of OSC messages in order mediate new values, and the synthesizer in the back-end need to handle these messages. OSC message passing is implemented so that when the user switches synth in the back-end, the GUI will respond to this, switch to the slider panel. The code for all this is found in listings A.12, A.13, A.14. Figure 4.5 shows the complete GUI.

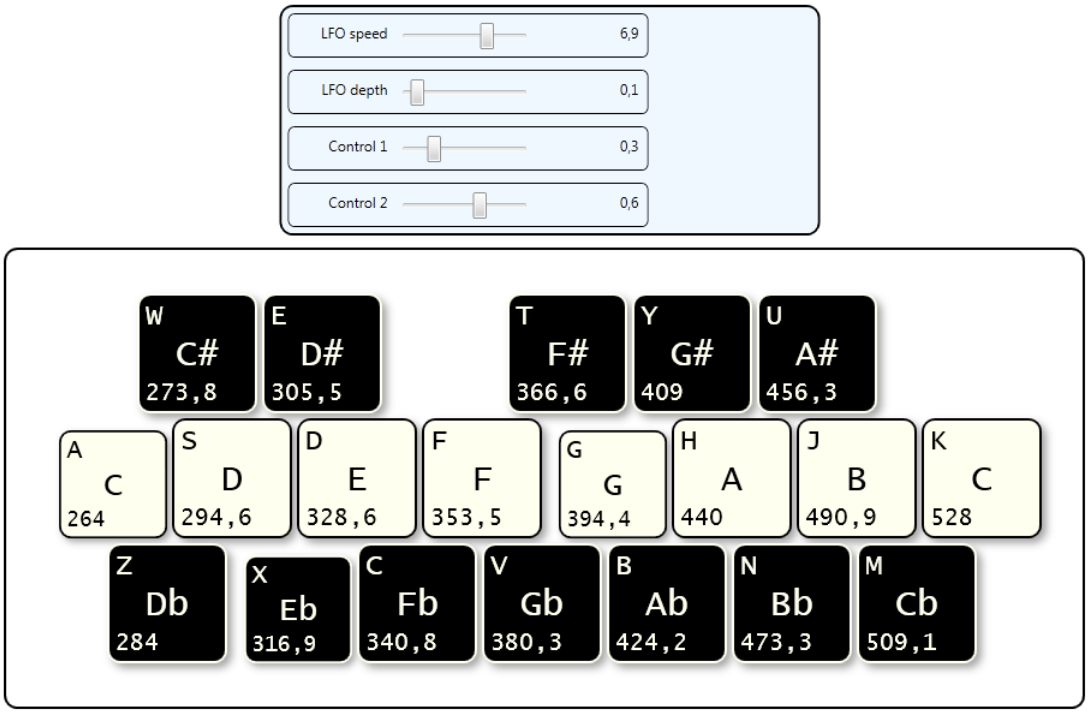

Figure 4.5. The user interface

Discussion of results and conclusions

The main aim within this project has been to develop a software tool for interactive demonstration of the *audible* differences between the musical tuning systems 12-TET and 19-TET, and not to evaluate the differences themselves. Therefore, I personally consider the actual software instrument to be the main result of this project. The GUI is very intuitive and responsive, and the really good thing about it is that by hitting the Enter-key, the synthesizer will re-tune from 12-TET to 19-TET or vice versa, and the keyboard layout will re-map itself, showing the keys, frequencies and note names corresponding to the current tuning. This definitely argues for its usefulness in any future comparative study of the tunings. The largest problem with the instrument is the lack of several, simultaneously mapped octaves. The layout as is now is limited to just one octave, which clearly is not sufficient to play anything serious. Switching octaves up and down is done be by pressing the up and down keys, but there is only one octave at the time.

To shortly address the matter of purity differences regarding the two tuning systems, it is, by using the instruments ability to quickly switch between tunings, very easy to confirm that the thirds and sixths in 19-TET indeed are noticeably purer than in 12-TET, these intervals are *really* pure. The loss of purity in the fifth interval in 19-TET is audible as well, but it really is not that bad.

Not being a trained keyboard player, I have to admit it is difficult to say whether the keyboard adaptation is the best possible one. Although, purely speculative, Yasser's keyboard, see figure 4.1. seems more practical, at least if you are used to a regular piano keyboard. But as discussed in the chapter Method this was never really a practical alternative.

Initially during the development of the synthesizer, awe was felt before the problem of managing voice allocation, but using ChucK's event system this problem disappeared. ChucK in general has been very easy to learn and use, although the language is not fully developed. For example, access modifiers, constructors and referencing variables of primitive data types, are all language features which are yet to be implemented. Aside from that there was no real problems to speak of. Using OSC to achieve inter-application communication was, considering the alternative of basically implement something similar from scratch, really easy once figured out.

Getting into the theoretical aspects of music has led to a widened perspective

towards music in general, and given knowledge that most definitely will be of use during contingent further studies in the field. If report writing had not taken as much time as it did, there might had been time for maybe using ChucK to compose a smaller musical piece for 19-TET. Maybe it would be a nice summer project.

Bibliography

[1] A440 (pitch standard). http://en.wikipedia.org/wiki/A440_(pitch_standard).

[2] Chromatic scale. http://en.wikipedia.org/wiki/Chromatic_scale.

[3] Equal temperament. http://en.wikipedia.org/wiki/Equal_temperament.

[4] Generalized keyboard. http://en.wikipedia.org/wiki/Generalized_keyboard.

[5] Harmonic series. http://en.wikipedia.org/wiki/Harmonic_series_(mathematics).

[6] Introduction to osc. http://opensoundcontrol.org/introduction-osc.

[7] Just intonation. http://en.wikipedia.org/wiki/Just_intonation.

[8] Open sound control api. http://www.bespokesoftware.org/wordpress/?page_id=69/.

[9] Pitch class. http://en.wikipedia.org/wiki/Pitch_class.

[10] Pitch (music). http://en.wikipedia.org/wiki/Pitch_(music).

[11] Xaml overview (wpf). http://msdn.microsoft.com/en-us/library/ms752059.aspx.

[12] Ivor Darreg. A case for nineteen. *INTERVAL - A Microtonal Newsletter*, 1979. http://sonic-arts.org/darreg/case.htm.

[13] Scott Hackleman. The hackleman-wilson 19-tone clavichord. In *Microfest 2001 - Conference and festival of music in alternative tunings*, 2001. http://www2.hmc.edu/ alves/microfestabstracts.html.

[14] Mayer Joel Mandelbaum. *Multiple division of the octave and the tonal resources of 19-tone temperament.* PhD thesis, Indiana University, 1961. http://anaphoria.com/mandelbaum.html.

[15] Erkki Huovinen Saku Bucht. Perceived consonance of harmonic intervals in 19-tone equal temperament. In F. Zimmer (Eds.) R. Parncutt, A. Kessler, editor, *Proceedings of the Conference on Interdisciplinary Musicology (CIM04) Graz/Austria, 15-18 April, 2004*, 2004. http://gewi.uni-graz.at/ cim04/.

[16] Erv Wilson. 19-tone scale for the clavichord-19. http://www.anaphoria.com/Clavichord19-17-22.pdf, 1976.

[17] Joseph Yasser. *A Theory of Evolving Tonality*. American Library of Musicology, 1932. http://www.musanim.com/Yasser.

Terms and acronyms

12-TET Twelve-tone Equal Temperament. 1–5, 9, 13, 15, 16, 19, 23

19-TET Nineteen-tone Equal Temperament. 2–6, 9, 11, 12, 15, 16, 19

API Application Programming Interface. 17

cent A logarithmic unit for measuring musical intervals. 1, 5, 6

chromatic scale The musical scale of all pitches in a given tuning. For example, in 12-TET, the chromatic scale is all pitches in 12-TET, each a semitone apart. 1

ChucK A programming language and a virtual machine for development of audio synthesis programs. 5–7, 9, 11, 14, 16, 17, 19, 24

GUI Graphical User Interface. 9, 11, 12, 15–18

harmonic A harmonic of a wave having a fundamental frequency is another wave with a frequency that is an integer multiple of the fundamental frequency. 2

harmonic series The infinite series of all harmonics of a wave given a fundamental frequency. 2

interval The combination of two notes, or the distance between their pitches. This distance is commonly expressed as a mathematical ratio between the frequencies of the two pitches. 1, 2, 5, 24

MIDI Musical Instrument Digital Interface, a protocol enabling electronic musical instruments to communicate. 11

OSC Open Sound Control, a light-weight text-based network protocol for communication between applications. 15–19

pitch A less formal and more music oriented synonym to the term frequency. 1, 23

pitch class A pitch class is the set of pitches that are a whole number of octaves apart. For instance, on a standard piano there are seven C keys, $C1, C2, ...C7$. These form the C pitch class. 1

semitone The smallest musical interval commonly used in western music. It is the interval between two adjacent notes in a twelve tone scale. 1, 23

shred A thread or process in ChucK. 16

tuning system A system that defines pitches and how they relate to each other. 1–5, 9, 19

WPF Windows Presentation Foundation, a framework for developing user interfaces. 17

XAML a declarative markup language used to simplify UI development in .NET applications. 17

Source code

This section provides most of the source code produced during the implementation of the synthesizer. The code is divided in two sections, one for the ChucK code, implementing the actual synthesizer, and one for the C# code, implementing the graphical user interface. Some parts of the C# code is not accounted for here, as much of it is very similar. The code is presented in listings, one for each source file. At this point the software is largely prototypical, which is why there is no downloadable executable. If anyone is interested in taking a closer look, please e-mail me for a full copy of the source.

A.1 Back-end ChucK code

A.1.1 KeyEvent.ck

Listing A.1. KeyEvent.ck

```
1 public class KeyEvent extends Event
2 {
3   int ascii;
4
5   fun float getFreq(Mapper @ m) {
6     return m.map[ascii];
7   }
8 }
```

A.1.2 ShredCollection.ck

Listing A.2. ShredCollection.ck

```
1 public class ShredCollection
2 {
3   Shred @ shreds[1024];
4   0 => int head;
5
6   fun void register(Shred @ shred) {
7     if (shreds[head] != null) {
8       Machine.remove(shreds[head].id());
9       null @=> shreds[head];
```

```
10      }
11
12      shred @=> shreds[head];
13      (head + 1) \% shreds.cap() => head;
14    }
15
16    fun void killAll() {
17      for (0 => int i; i < head; i++)
18        if (shreds[i] != null)
19          Machine.remove(shreds[i].id());
20        0 => head;
21    }
22  }
```

A.1.3 OSCHelper.ck

Listing A.3. OSCHelper.ck

```
1  public class OSCHelper
2  {
3    4712 => static int listeningPort;
4    4710 => static int sendingPort;
5
6    OscRecv recv;
7    listeningPort => recv.port;
8    recv.listen();
9
10   OscSend send;
11   send.setHost("localhost", sendingPort);
12 }
```

A.1.4 Mapper.ck

Listing A.4. Mapper.ck

```
1  public class Mapper
2  {
3    OSCHelper @ osc;
4
5    19 => int numDivisions;
6    3 => int octave;
7    0 => int lowOctave;
8    6 => int highOctave;
9
10   //27.5625 => float freqA0;
11   27.5 => float freqA0;
12
13   computeFrequencies(0, 5, 19) @=> float freqs19[];
14   computeFrequencies(0, 3, 12) @=> float freqs12[];
15
16   [65, 87, 83, 69, 68, 70, 84, 71, 89, 72, 85, 74, 75] @=> int keys12[];
17   [65, 87, 90, 83, 69, 88, 68, 67, 70, 84, 86,
18    71, 89, 66, 72, 85, 78, 74, 77, 75] @=> int keys19[];
19
```

```
20    float map[256];
21
22    remap();
23
24    fun void setupKeys() {
25      reset(map);
26
27      octave * numDivisions => int octaveOffset;
28
29      if (numDivisions == 19) {
30        for (0 => int keyIdx; keyIdx < keys19.cap(); keyIdx++)
31          freqs19[keyIdx + octaveOffset] => map[keys19[keyIdx]];
32
33      } else if (numDivisions == 12) {
34        for (0 => int keyIdx; keyIdx < keys12.cap(); keyIdx++)
35          freqs12[keyIdx + octaveOffset] => map[keys12[keyIdx]];
36      }
37    }
38
39    fun float[] computeFrequencies(int octaveIdx,
40                                   float startPitchIdx,
41                                   int numDiv) {
42      freqA0 * Math.pow(2, octaveIdx) => float octaveFreq;
43      Math.pow(2, startPitchIdx / (numDiv \$ float))
44        => float startPitchRatio;
45      octaveFreq * startPitchRatio => float startPitchFreq;
46      float freqs[(highOctave + 1) * numDiv + 1];
47
48      for (0 => int pitchIdx; pitchIdx < freqs.cap(); pitchIdx++) {
49        Math.pow(2, pitchIdx / (numDiv \$ float)) * startPitchFreq =>
50          float semiToneFreq;
51        semiToneFreq => freqs[pitchIdx];
52      }
53
54      return freqs;
55    }
56
57    fun static void reset(float map[]) {
58      for (0 => int i; i < map.cap(); i++) 0 => map[i];
59    }
60
61    fun void sendMapping() {
62      if (osc == null)
63        return;
64
65      if (numDivisions == 19) {
66        osc.send.startMsg("/func/toggle_tuning,i");
67        osc.send.addInt(numDivisions);
68        osc.send.startMsg("/func/mapping,i␣f␣i␣f␣i␣f␣i
69  ␣␣␣␣␣␣␣␣f␣i␣f␣i␣f␣i␣f␣i␣f␣i␣f␣i␣f␣i␣f␣i␣f␣i␣f␣i␣f␣i␣f␣i␣f␣i␣f␣i␣f␣i␣f␣i␣f␣i␣f");
70        for (0 => int i; i <= 19; i++) {
71          osc.send.addInt(keys19[i]);
72          osc.send.addFloat(map[keys19[i]]);
73        }
```

```
74      } else if (numDivisions == 12) {
75        osc.send.startMsg("/func/toggle_tuning,i");
76        osc.send.addInt(numDivisions);
77        osc.send.startMsg("/func/mapping,i␣f␣i␣f␣i␣f␣i
78  ␣␣␣␣␣␣␣␣f␣i␣f␣i␣f␣i␣f␣i␣f␣i␣f␣i␣f␣i␣f␣i␣f");
79        for (0 => int i; i <= 12; i++) {
80          osc.send.addInt(keys12[i]);
81          osc.send.addFloat(map[keys12[i]]);
82        }
83      }
84    }
85
86    fun void remap() {
87      <<< "switching␣to␣", numDivisions + "-TET" >>>;
88      setupKeys();
89      sendMapping();
90    }
91
92    fun void remap(int numDiv) {
93      numDiv => numDivisions;
94      remap();
95    }
96  }
```

A.1.5 AbstractPolySynth.ck

Listing A.5. AbstractPolySynth.ck

```
1   public class AbstractPolySynth
2   {
3     UGen @ voices[];
4     UGen @ output;
5     ShredCollection shreds;
6     OSCHelper @ osc;
7     Mapper @ mapper;
8     KeyEvent @ keyDown;
9     KeyEvent @ keyUp;
10    int numVoices;
11    int noteOnOnly;
12
13    fun pure ShredCollection@ getShreds();
14    fun pure void init(int numVoices,
15                       int noteOnOnly,
16                       Mapper@ mapper,
17                       KeyEvent@ kDown,
18                       KeyEvent@ kUp);
19    fun pure void voiceHandler(int id);
20    fun pure void start();
21
22    fun void stop() {
23      shreds.killAll();
24    }
25
26    fun void disconnect() {
```

```
27      for (0 => int i; i < numVoices; i++)
28        voices[i] =< output;
29    }
30
31    fun void connect(UGen @ out) {
32      out @=> output;
33      for (0 => int i; i < numVoices; i++)
34        voices[i] => output;
35    }
36
37    fun void pingHandler() {
38      shreds.register(me);
39      osc.recv.event("/ping") @=> OscEvent pingEvent;
40      while (true) {
41        <<< "waiting for GUI to ping", "" >>>;
42        pingEvent => now;
43        <<< "got ping from GUI", "" >>>;
44        while (pingEvent.nextMsg() != 0);
45        mapper.remap(19);
46      }
47    }
48
49    fun void noteOnHandler() {
50      shreds.register(me);
51      osc.recv.event("/note/on,i") @=> OscEvent noteOnEvent;
52      while (true) {
53        noteOnEvent => now;
54        while (noteOnEvent.nextMsg() != 0) {
55          noteOnEvent.getInt() => int ascii;
56
57          if (mapper.map[ascii] != 0) {
58            ascii => keyDown.ascii;
59            keyDown.signal();
60          }
61        }
62      }
63    }
64
65    fun void noteOffHandler() {
66      shreds.register(me);
67      osc.recv.event("/note/off,i") @=> OscEvent noteOffEvent;
68      while (true) {
69        noteOffEvent => now;
70        while (noteOffEvent.nextMsg() != 0) {
71          noteOffEvent.getInt() => int ascii;
72
73          if (mapper.map[ascii] != 0) {
74            ascii => keyUp.ascii;
75            keyUp.signal();
76          }
77        }
78      }
79    }
80
```

```
81    fun void startBase() {
82      for (0 => int i; i < numVoices; i++)
83        spork ~ voiceHandler(i);
84
85      spork ~ pingHandler();
86      spork ~ noteOnHandler();
87      spork ~ noteOffHandler();
88    }
89  }
```

A.1.6 PolyFMVoices.ck

Listing A.6. PolyFMVoices.ck

```
1   public class PolyFMVoices extends AbstractPolySynth
2   {
3     fun void init(int numVoices,
4                   int noteOnOnly,
5                   Mapper@ m,
6                   KeyEvent@ keyDown,
7                   KeyEvent@ keyUp) {
8       numVoices => this.numVoices;
9       noteOnOnly => this.noteOnOnly;
10      keyDown @=> this.keyDown;
11      keyUp @=> this.keyUp;
12      m @=> mapper;
13      new UGen[numVoices] @=> voices;
14      for (0 => int i; i < numVoices; i++) {
15        FMVoices fmv @=> voices[i];
16        1 => fmv.noteOff;
17        0.2 => fmv.gain;
18      }
19    }
20
21    fun void voiceHandler(int id) {
22      shreds.register(me);
23      while (true) {
24        keyDown => now;
25        keyDown.getFreq(mapper) => (voices[id] \$ BeeThree).freq;
26        1 => (voices[id] \$ FM).noteOn;
27        while (!noteOnOnly) {
28          keyUp => now;
29          1 => (voices[id] \$ FM).noteOff;
30          break;
31        }
32      }
33    }
34
35    fun void vowelHandler() {
36      shreds.register(me);
37      osc.recv.event("/func/vowel,f") @=> OscEvent vowelEvent;
38      while (true) {
39        vowelEvent => now;
40        while (vowelEvent.nextMsg() != 0) {
```

```
41          vowelEvent.getFloat() => float vowel;
42          for (0 => int i; i < numVoices; i++)
43            vowel => (voices[i] \$ FMVoices).vowel;
44        }
45      }
46    }
47
48    fun void spectralTiltHandler() {
49      shreds.register(me);
50      osc.recv.event("/func/spectraltilt,f") @=> OscEvent spectralTiltEvent;
51      while (true) {
52        spectralTiltEvent => now;
53        while (spectralTiltEvent.nextMsg() != 0) {
54          spectralTiltEvent.getFloat() => float tilt;
55          for (0 => int i; i < numVoices; i++)
56            tilt => (voices[i] \$ FMVoices).spectralTilt;
57        }
58      }
59    }
60
61    fun void start() {
62      startBase();
63      spork ~ vowelHandler();
64      spork ~ spectralTiltHandler();
65    }
66  }
```

A.1.7 main.ck

Listing A.7. main.ck

```
1  KeyEvent keyDown;
2  KeyEvent keyUp;
3  OSCHelper osc;
4
5  Mapper mapper;
6  osc @=> mapper.osc;
7
8  PolyBeeThree organ;
9  organ.init(16, 0, mapper, keyDown, keyUp);
10 osc @=> organ.osc;
11
12 PolyFMVoices fmVoices;
13 fmVoices.init(16, 0, mapper, keyDown, keyUp);
14 osc @=> fmVoices.osc;
15
16 AbstractPolySynth synths[2];
17 AbstractPolySynth @ currentSynth;
18
19 organ @=> synths[0];
20 fmVoices @=> synths[1];
21
22 organ @=> currentSynth;
23
```

```
24 0 => int synthIdx;
25
26 synths[0].start();
27 synths[0].connect(dac);
28
29 fun void mainProgram() {
30   Hid hi;
31   HidMsg msg;
32
33   0 => int device;
34   //if( me.args() ) me.arg(0) => Std.atoi => device;
35   if(!hi.openKeyboard(device))
36     me.exit();
37   <<< "keyboard '" + hi.name() + "' ready", "" >>>;
38
39   while(true) {
40     hi => now;
41     while(hi.recv(msg)) {
42       if (msg.isButtonDown()) {
43         // left arrow key --> toggle synth!
44         if (msg.which == 203) {
45           synths[synthIdx].stop();
46           synths[synthIdx].disconnect();
47           (synthIdx + 1) % 2 => synthIdx;
48           synths[synthIdx].start();
49           synths[synthIdx].connect(dac);
50           synths[synthIdx] @=> currentSynth;
51
52           osc.send.startMsg("/func/loadsynth,i");
53           osc.send.addInt(synthIdx);
54         } else
55
56         // up arrow key --> shift octave upwards
57         if (msg.which == 200) {
58           mapper.octave < mapper.highOctave ?
59             mapper.octave + 1: mapper.highOctave => mapper.octave;
60           mapper.remap();
61         // down arrow key --> shift octave downwards
62         } else if (msg.which == 208) {
63           mapper.octave > mapper.lowOctave ?
64             mapper.octave - 1: mapper.lowOctave => mapper.octave;
65           mapper.remap();
66         // return key --> toggle 12-TET/19-TET
67         } else if (msg.ascii == 10) {
68           if (mapper.numDivisions == 19)
69             mapper.remap(12);
70           else if (mapper.numDivisions == 12)
71             mapper.remap(19);
72         }
73         // any of the keyboard mapped keys down --> note on
74         else if (mapper.map[msg.ascii] != 0) {
75           msg.ascii => keyDown.ascii;
76           keyDown.signal();
77           osc.send.startMsg("/note/on,i");
```

```
78              osc.send.addInt(msg.ascii);
79            }
80          // key up
81          } else {
82            // any of the keyboard mapped keys up --> note off
83            if (mapper.map[msg.ascii] != 0) {
84              msg.ascii => keyUp.ascii;
85              keyUp.signal();
86              osc.send.startMsg("/note/off,i");
87              osc.send.addInt(msg.ascii);
88            }
89          }
90        }
91      }
92    }
93
94    mainProgram();
```

A.2 UI C#/.NET code

A.2.1 OSC.cs

Listing A.8. OSC.cs

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using Bespoke.Common.Osc;
6   using System.Net;
7
8   namespace SynTET
9   {
10    static class OSC
11    {
12      public static readonly int SourcePort = 4711;
13      public static readonly int DestinationPort = 4712;
14
15      public static readonly IPEndPoint DestinationEndPoint =
16        new IPEndPoint(IPAddress.Loopback, DestinationPort);
17      public static readonly IPEndPoint SourceEndPoint =
18        new IPEndPoint(IPAddress.Loopback, SourcePort);
19
20      public static readonly OscMessage NoteOn =
21        new OscMessage(SourceEndPoint, "/note/on");
22      public static readonly OscMessage NoteOff =
23        new OscMessage(SourceEndPoint, "/note/off");
24      public static readonly OscMessage ToggleTuning =
25        new OscMessage(SourceEndPoint, "/func/toggle_tuning");
26      public static readonly OscMessage LoadSynth =
27        new OscMessage(SourceEndPoint, "/func/loadsynth");
28      public static readonly OscMessage Mapping =
29        new OscMessage(SourceEndPoint, "/func/mapping");
30      public static readonly OscMessage Ping =
```

```
31        new OscMessage(SourceEndPoint, "/ping");
32      public static readonly OscMessage Pong =
33        new OscMessage(SourceEndPoint, "/pong");
34
35
36      public static readonly OscMessage Vowel =
37        new OscMessage(SourceEndPoint, "/func/vowel");
38      public static readonly OscMessage SpectralTilt =
39        new OscMessage(SourceEndPoint, "/func/spectraltilt");
40
41      public static readonly OscMessage LFOSpeed =
42        new OscMessage(SourceEndPoint, "/func/lfo/speed");
43      public static readonly OscMessage LFODepth =
44        new OscMessage(SourceEndPoint, "/func/lfo/depth");
45      public static readonly OscMessage ControlOne =
46        new OscMessage(SourceEndPoint, "/func/control1");
47      public static readonly OscMessage ControlTwo =
48        new OscMessage(SourceEndPoint, "/func/control2");
49    }
50  }
```

A.2.2 Server.cs

Listing A.9. Server.cs

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using Bespoke.Common.Osc;
6   using System.Net;
7   using System.Windows.Threading;
8   using System.Windows.Controls;
9
10  namespace SynTET
11  {
12    class Server
13    {
14      private readonly OscServer server_;
15      private static readonly int port_ = 4710;
16      private static readonly IPAddress address_ = IPAddress.Loopback;
17      private readonly Keyboard keyboard_;
18      private readonly StackPanel panel_;
19
20      public Server(MainWindow mainWindow) {
21        keyboard_ = mainWindow.keyboard_;
22
23        server_ = new OscServer(TransportType.Udp,
24                                IPAddress.Loopback,
25                                port_,
26                                address_,
27                                Bespoke.Common.Net.TransmissionType.LocalBroadcast);
28        server_.RegisterMethod(OSC.NoteOn.Address);
29        server_.RegisterMethod(OSC.NoteOff.Address);
```

```
30          server_.RegisterMethod(OSC.ToggleTuning.Address);
31          server_.RegisterMethod(OSC.Mapping.Address);
32          server_.RegisterMethod(OSC.LoadSynth.Address);
33
34          server_.MessageReceived += (o, e) => {
35            if (e.Message.Address.Substring(0, 5).
36                  Equals(OSC.NoteOff.Address.Substring(0, 5))) {
37              bool k = false;
38              if (e.Message.Address.Equals(OSC.NoteOn.Address))
39                k = true;
40              else if (e.Message.Address.Equals(OSC.NoteOff.Address))
41                k = false;
42
43              keyboard_.Dispatcher.
44                BeginInvoke(keyboard_.UpdateButton, e.Message.At<int>(0), k);
45            } else if (e.Message.Address.Equals(OSC.ToggleTuning.Address)) {
46              keyboard_.Dispatcher.
47                BeginInvoke(keyboard_.ToggleKeyboard, e.Message.At<int>(0));
48            } else if (e.Message.Address.Equals(OSC.Mapping.Address)) {
49              keyboard_.Dispatcher.BeginInvoke(keyboard_.UpdatePitches, e.Message);
50            } else if (e.Message.Address.Equals(OSC.LoadSynth.Address)) {
51              mainWindow.Dispatcher.
52                BeginInvoke(mainWindow.ChangePanel, e.Message.At<int>(0));
53            }
54          };
55
56          server_.Start();
57          keyboard_.Dispatcher.BeginInvoke(keyboard_.Ping, null);
58        }
59      }
60  }
```

A.2.3 Keyboard.xaml.cs

Listing A.10. Keyboard.xaml.cs

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using System.Windows;
6   using System.Windows.Controls;
7   using System.Windows.Data;
8   using System.Windows.Documents;
9   using System.Windows.Input;
10  using System.Windows.Media;
11  using System.Windows.Media.Imaging;
12  using System.Windows.Navigation;
13  using System.Windows.Shapes;
14  using Bespoke.Common.Osc;
15  using System.Net;
16  using System.Windows.Threading;
17
18  namespace SynTET
```

```
19 {
20   public partial class Keyboard : UserControl
21   {
22     private readonly KeyControl[] keyboard_ = new KeyControl[256];
23
24     public Action<int, bool> UpdateButton;
25     public Action<int> ToggleKeyboard;
26     public Action<OscMessage> UpdatePitches;
27     public Action<KeyControl> SendNoteOn;
28     public Action<KeyControl> SendNoteOff;
29     public Action Ping;
30
31     public Keyboard()
32     {
33       UpdatePitches = new Action<OscMessage>(msg => {
34         for (int i = 0; i < msg.Data.Length; i += 2)
35           keyboard_[msg.At<int>(i)].Frequency =
36             String.Format("{0:0.#}", msg.At<float>(i + 1));
37       });
38
39       SendNoteOn = new Action<KeyControl>(kc => {
40         OSC.NoteOn.Append<int>((int)kc.ASCII[0]);
41         OSC.NoteOn.Send(OSC.DestinationEndPoint);
42         OSC.NoteOn.ClearData();
43       });
44
45       SendNoteOff = new Action<KeyControl>(kc => {
46         OSC.NoteOff.Append<int>((int)kc.ASCII[0]);
47         OSC.NoteOff.Send(OSC.DestinationEndPoint);
48         OSC.NoteOff.ClearData();
49       });
50
51       UpdateButton = new Action<int, bool>((idx, mode) => {
52         if (keyboard_[idx] != null) keyboard_[idx].IsDown = mode;
53       });
54
55       ToggleKeyboard = new Action<int>((tet) => {
56         if (tet == 12) {
57           keyboard_['W'].PitchClass = "C#/Db";
58           keyboard_['E'].PitchClass = "D#/Eb";
59           keyboard_['T'].PitchClass = "F#/Gb";
60           keyboard_['Y'].PitchClass = "G#/Ab";
61           keyboard_['U'].PitchClass = "A#/Bb";
62           stackPanel2_.Visibility = Visibility.Hidden;
63         } else if (tet == 19) {
64           keyboard_['W'].PitchClass = "C#";
65           keyboard_['E'].PitchClass = "D#";
66           keyboard_['T'].PitchClass = "F#";
67           keyboard_['Y'].PitchClass = "G#";
68           keyboard_['U'].PitchClass = "A#";
69
70           stackPanel2_.Visibility = Visibility.Visible;
71         }
72       });
```

```
73
74          Ping = new Action(() => {
75            OSC.Ping.Send(OSC.DestinationEndPoint);
76          });
77
78          InitializeComponent();
79
80          // code for this method is omitted
81          InitializeKeys();
82        }
83    }
84  }
```

A.2.4 KeyControl.xaml.cs

Listing A.11. KeyControl.xaml.cs

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using System.Windows;
6   using System.Windows.Controls;
7   using System.Windows.Data;
8   using System.Windows.Documents;
9   using System.Windows.Input;
10  using System.Windows.Media;
11  using System.Windows.Media.Imaging;
12  using System.Windows.Navigation;
13  using System.Windows.Shapes;
14  using System.Windows.Controls.Primitives;
15  using System.ComponentModel;
16  using Bespoke.Common.Osc;
17
18  namespace SynTET
19  {
20    public partial class KeyControl : UserControl
21    {
22      // dependency property initializers are omitted
23
24      private readonly Keyboard keyboard_;
25
26      public KeyControl(Keyboard kbd, string ascii, string notename) {
27        InitializeComponent();
28        keyboard_ = kbd;
29
30        MouseDown += (o, e) => {
31          IsDown = true;
32          keyboard_.Dispatcher.BeginInvoke(keyboard_.SendNoteOn, this);
33        };
34
35        MouseUp += (o, e) => {
36          IsDown = false;
37          keyboard_.Dispatcher.BeginInvoke(keyboard_.SendNoteOff, this);
```

```
38          };
39
40            ASCII = ascii;
41            PitchClass = notename;
42          }
43
44          public KeyControl(Keyboard kbd, string ascii, string notename,
45              SolidColorBrush black, SolidColorBrush white)
46            : this(kbd, ascii, notename) {
47            Black = black;
48            White = white;
49          }
50
51        // property accessors are omitted
52      }
53  }
```

A.2.5 MainWindow.xaml.cs

Listing A.12. MainWindow.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Bespoke.Common.Osc;
using System.Net;
using System.Windows.Controls.Primitives;
using System.Windows.Threading;

namespace SynTET
{
  public partial class MainWindow : Window
  {
    public Action<int> ChangePanel;

    public MainWindow()
    {
      InitializeComponent();
      ChangePanel = new Action<int>((idx) =>
      {
        stackPanel_.Children.RemoveAt(0);

        if (idx == 0)
        stackPanel_.Children.Insert(0, new BeeThreePanel());
```

```
        else if (idx == 1)
        stackPanel_.Children.Insert(0, new FMVoicesPanel());
      });

      Server server = new Server(this);
    }
  }
}
```

A.2.6 SliderControl.xaml.cs

Listing A.13. SliderControl.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace SynTET
{
  public partial class SliderControl : UserControl
  {
    public event Action<float> Move;
    // dependency property definitions omitted
    // property accessors omitted
    public SliderControl()
    {
      InitializeComponent();
    }

    private void Slider_MouseMove(object sender, MouseEventArgs e)
    {
      if (e.LeftButton == MouseButtonState.Pressed)
      {
        double value = ((Slider)sender).Value;
        Value = value;

        if (Move != null)
          Move((float)value);
       }
     }
  }
}
```

A.2.7 FMVoicesPanel.xaml.cs

Listing A.14. FMVoicesPanel.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace SynTET
{
  public partial class FMVoicesPanel : UserControl
  {
    public FMVoicesPanel()
    {
      InitializeComponent();

      vowel_.Move += (value) =>
      {
        OSC.Vowel.Append<float>((float)value);
        OSC.Vowel.Send(OSC.DestinationEndPoint);
        OSC.Vowel.ClearData();
      };

      spectralTilt_.Move += (value) =>
      {
        OSC.SpectralTilt.Append<float>((float)value);
        OSC.SpectralTilt.Send(OSC.DestinationEndPoint);
        OSC.SpectralTilt.ClearData();
      };

      adsrTarget_.Move += (value) =>
      {
        OSC.ADSRTarget.Append<float>((float)value);
        OSC.ADSRTarget.Send(OSC.DestinationEndPoint);
        OSC.ADSRTarget.ClearData();
      };

    }
  }
}
```

A.3 UI XAML code

A.3.1 Keyboard.xaml

Listing A.15. Keyboard.xaml

```
<UserControl x:Class="SynTET.Keyboard"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  xmlns:my="clr-namespace:SynTET"
  mc:Ignorable="d" Margin="5" Width="900">
  <Border BorderBrush="Black" BorderThickness="2" CornerRadius="10">
    <Border.Background>
      <SolidColorBrush Color="Transparent" />
    </Border.Background>
    <Grid Width="880" Height="380">
      <Grid.RowDefinitions>
        <RowDefinition />
        <RowDefinition Height="auto" />
        <RowDefinition Height="auto" />
        <RowDefinition Height="auto" />
        <RowDefinition />
      </Grid.RowDefinitions>
      <Grid.ColumnDefinitions>
        <ColumnDefinition />
        <ColumnDefinition Width="auto" />
        <ColumnDefinition />
      </Grid.ColumnDefinitions>
      <StackPanel Grid.Column="1" Grid.Row="1"
        Orientation="Horizontal" Name="stackPanel0_" />
      <StackPanel Grid.Column="1" Grid.Row="2"
        Orientation="Horizontal" Name="stackPanel1_" />
      <StackPanel Grid.Column="1" Grid.Row="3"
        Orientation="Horizontal" Name="stackPanel2_" />
    </Grid>
  </Border>
</UserControl>
```

A.3.2 KeyControl.xaml

Listing A.16. KeyControl.xaml

```
1  <UserControl x:Class="SynTET.KeyControl"
2    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
5    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
6    xmlns:my="clr-namespace:SynTET"
7    mc:Ignorable="d" d:DesignHeight="200" d:DesignWidth="200">
8    <UserControl.Resources>
9      <Storyboard x:Key="scaleDown">
10       <DoubleAnimation Storyboard.TargetName="scaleTransform_"
11         Duration="00:00:00:0.05" Storyboard.TargetProperty="ScaleX"
12         From="1" To="0.9" />
13       <DoubleAnimation Storyboard.TargetName="scaleTransform_"
14         Duration="00:00:00:0.05" Storyboard.TargetProperty="ScaleY"
15         From="1" To="0.9" />
16     </Storyboard>
```

```
17      <Storyboard x:Key="translateDown">
18        <DoubleAnimation Storyboard.TargetName="translateTransform_"
19          Duration="00:00:00:0.03" Storyboard.TargetProperty="X"
20          From="0" To="5" />
21        <DoubleAnimation Storyboard.TargetName="translateTransform_"
22          Duration="00:00:00:0.03" Storyboard.TargetProperty="Y"
23          From="0" To="5" />
24      </Storyboard>
25      <Storyboard x:Key="scaleUp">
26        <DoubleAnimation Storyboard.TargetName="scaleTransform_"
27          Duration="00:00:00:0.05" Storyboard.TargetProperty="ScaleX"
28          From="0.9" To="1" />
29        <DoubleAnimation Storyboard.TargetName="scaleTransform_"
30          Duration="00:00:00:0.05" Storyboard.TargetProperty="ScaleY"
31          From="0.9" To="1" />
32      </Storyboard>
33      <Storyboard x:Key="translateUp">
34        <DoubleAnimation Storyboard.TargetName="translateTransform_"
35          Duration="00:00:00:0.03" Storyboard.TargetProperty="X"
36          From="5" To="0" />
37        <DoubleAnimation Storyboard.TargetName="translateTransform_"
38          Duration="00:00:00:0.03" Storyboard.TargetProperty="Y"
39          From="5" To="0" />
40      </Storyboard>
41    </UserControl.Resources>
42
43    <UserControl.Template>
44      <ControlTemplate>
45        <Canvas Width="100" Height="100" Background="Transparent" Margin="2">
46          <Border Name="border_"  CornerRadius="10" Width="100" Height="100"
47            BorderThickness="2" BorderBrush="{TemplateBinding␣my:KeyControl.Black}"
48            Background="{TemplateBinding␣my:KeyControl.White}" >
49            <Border.Effect>
50              <DropShadowEffect BlurRadius="10" Color="LightGray"
51                RenderingBias="Performance" />
52            </Border.Effect>
53            <Border.RenderTransform>
54              <TransformGroup>
55                <ScaleTransform x:Name="scaleTransform_" CenterX="50" CenterY="50" />
56                <TranslateTransform x:Name="translateTransform_" />
57              </TransformGroup>
58            </Border.RenderTransform>
59            <StackPanel Orientation="Vertical">
60              <TextBlock Name="ascii_" Margin="5,5,0,5"
61                Text="{TemplateBinding␣my:KeyControl.ASCII}" HorizontalAlignment="Left"
62                VerticalAlignment="Top"  FontFamily="Lucida␣Console" FontSize="24"
63                Foreground="{Binding␣ElementName=border_,␣Path=BorderBrush}" />
64              <TextBlock Name="pitch_" Text="{TemplateBinding␣my:KeyControl.PitchClass}"
65                HorizontalAlignment="Center" VerticalAlignment="Center"
66                FontFamily="Lucida␣Console" FontSize="30"
67                Foreground="{Binding␣ElementName=border_,␣Path=BorderBrush}" />
68              <TextBlock Name="freq_"  Margin="5,7,0,0"
69                Text="{TemplateBinding␣my:KeyControl.Frequency}"
70                HorizontalAlignment="Left" VerticalAlignment="Bottom"
```

```
71                  FontFamily="Lucida Console" FontSize="20"
72                  Foreground="{Binding ElementName=border_, Path=BorderBrush}"/>
73              </StackPanel>
74            </Border>
75          </Canvas>
76          <ControlTemplate.Triggers>
77            <Trigger Property="my:KeyControl.IsDown" Value="True">
78              <Trigger.EnterActions>
79                <BeginStoryboard Storyboard="{StaticResource ResourceKey=scaleDown}" />
80                <BeginStoryboard Storyboard="{StaticResource ResourceKey=translateDown}" />
81              </Trigger.EnterActions>
82              <Trigger.ExitActions>
83                <BeginStoryboard Storyboard="{StaticResource ResourceKey=scaleUp}" />
84                <BeginStoryboard Storyboard="{StaticResource ResourceKey=translateUp}" />
85              </Trigger.ExitActions>
86            </Trigger>
87          </ControlTemplate.Triggers>
88        </ControlTemplate>
89      </UserControl.Template>
90  </UserControl>
```

A.3.3 MainWindow.xaml

Listing A.17. MainWindow.xaml

```
<Window x:Class="SynTET.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:my="clr-namespace:SynTET"
  Title="MainWindow" Width="935" Height="635" Margin="5"
  Closed="Window_Closed" >
  <StackPanel Orientation="Vertical" Name="stackPanel_" >
    <my:BeeThreePanel x:Name="sliderGroup_" HorizontalAlignment="Left"/>
    <my:Keyboard x:Name="keyboard_" HorizontalAlignment="Left" />
  </StackPanel>
</Window>
```

A.3.4 SliderControl.xaml

Listing A.18. SliderControl.xaml

```
<UserControl x:Class="SynTET.SliderControl"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  xmlns:my="clr-namespace:SynTET"
  mc:Ignorable="d"
  d:DesignHeight="50" d:DesignWidth="250" Margin="5" Width="300" Height="37">
  <UserControl.Resources>
    <Style x:Key="sliderStyle0" TargetType="{x:Type Slider}">
      <Setter Property="Margin" Value="0,7,7,0" />
    </Style>
```

```
    <Style x:Key="textBlockStyle0" TargetType="{x:Type␣TextBlock}">
      <Setter Property="HorizontalAlignment" Value="Right" />
      <Setter Property="Margin" Value="0,7,7,0" />
    </Style>
  </UserControl.Resources>
  <UserControl.Template>
    <ControlTemplate>
      <Border BorderBrush="Black" BorderThickness="1" CornerRadius="5">
        <Border.Background>
          <SolidColorBrush Color="AliceBlue" />
        </Border.Background>

        <Grid>
          <Grid.ColumnDefinitions>
            <ColumnDefinition />
            <ColumnDefinition Width="120" />
            <ColumnDefinition />
          </Grid.ColumnDefinitions>
          <Grid.RowDefinitions>
            <RowDefinition />
          </Grid.RowDefinitions>

          <TextBlock Grid.Row="0" Grid.Column="0"
            Text="{TemplateBinding␣my:SliderControl.Text}"
            Style="{StaticResource␣ResourceKey=textBlockStyle0}"/>
          <Slider Grid.Row="0" Grid.Column="1"
            Style="{StaticResource␣ResourceKey=sliderStyle0}"
            Maximum="{TemplateBinding␣my:SliderControl.Maximum}"
            Minimum="{TemplateBinding␣my:SliderControl.Minimum}"
            Value="{TemplateBinding␣my:SliderControl.Value}"
            MouseMove="Slider_MouseMove"/>
          <TextBlock Grid.Row="0" Grid.Column="2"
            Text="{TemplateBinding␣my:SliderControl.ValueText}"
            Style="{StaticResource␣ResourceKey=textBlockStyle0}" />
        </Grid>
      </Border>
    </ControlTemplate>
  </UserControl.Template>
</UserControl>
```

A.3.5 FMVoicesPanel.xaml

Listing A.19. FMVoicesPanel.xaml

```
<UserControl x:Class="SynTET.FMVoicesPanel"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  xmlns:my="clr-namespace:SynTET"
  mc:Ignorable="d"
  Margin="5" Width="450" Background="Transparent">
  <Border BorderBrush="Black" BorderThickness="2" CornerRadius="10">
    <Border.Background>
```

```
      <SolidColorBrush Color="AliceBlue" />
    </Border.Background>
    <StackPanel HorizontalAlignment="Left">
      <my:SliderControl x:Name="vowel_" Text="Vowel"
        Minimum="0" Maximum="1" Value="0.5" />
      <my:SliderControl x:Name="spectralTilt_" Text="Spectral Tilt"
        Minimum="0" Maximum="1" Value="0.5" />
      <my:SliderControl x:Name="adsrTarget_" Text="ADSR Target"
        Minimum="0" Maximum="1" Value="0.5" />
    </StackPanel>
  </Border>
</UserControl>
```