UNIVERSITÀ DI PADOVA ∽ FACOLTÀ DI INGEGNERIA
DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

TESI DI LAUREA SPECIALISTICA IN INGEGNERIA INFORMATICA

A practical approach to music theory on the Reactable

Andrea Franceschini

RELATORE
Prof. Giovanni De Poli

SUPERVISORE
Dott. Sergi Jordà

– The glory of creation is in its infinite diversity
– And the way our differences combine to create meaning and beauty
Dr. Miranda Jones and Spock, "Is there in truth no beauty?", star date 5630.8

Contents

Introduction

With the name "interactive music production systems" we identify those systems for music production, either musical instruments or other musical tools, based on computers, having most different interfaces, and requiring interaction with one or more users in order to fulfil their purpose. This is a rather general description that includes many different systems, from electronic musical instruments to sequencing and composition tools. The work hereafter presented is focused on electronic musical instruments based on tabletop and tangible interaction paradigms, using the Reactable as a case study and development platform.

In Chapter 1 we'll see the concept of audiovisual performance and we'll have a brief survey about systems that have been designed throughout history for this kind of performance, from early experiments to modern times, ending with a quick yet not too shallow description of the Reactable itself and its interaction paradigms. We'll also detail the motivations for this original work and we'll see a perspective on music education, and formulate an hypothesis about how the Reactable, together with the work here presented, may offer valuable help in the field.

Chapter 2 details the original work presented in this thesis. In this chapter we shall see all the iterations of the design process, from the first rough ideas to the final proposal, ready for assessment. We'll go through each step in the process, explaining in great detail how every single idea was conceived, evaluated and eventually included in the final proposal, or rejected.

Chapter 3 will present the demo application that was developed, also describing implementation details and practical choices.

Finally, in Chapter 4 we'll examine results and findings, as well as a number of future development proposals, both regarding enhancement that the implementation may include, and regarding music education and the use that can be made of this work by people with disabilities.

Disclaimer

This work has been done at the Music Technology Group of Universitat Pompeu Fabra in Barcelona under supervision of prof. Sergi Jordà. The results here presented reflect my personal research and opinions on the topic and, although being done in collaboration with the creators of the Reactable, they don't necessarily express the opinions and visions of the original team. Therefore all the developments I propose in this work are to be considered as a possible direction to be thoroughly examined, evaluated and validated through experimentation involving potential users of the system.

The latest version of this work will be available at

http://www.morpheu5.net/public/master_thesis/thesis.pdf

1

Background

What stranger enterprise could be imagined in the whole field of art than to make sound visible, to make available to the eyes those many pleasures which Music affords to the ears?
Louis-Bertrand Castel (1688-1757)

1.1 Audiovisual performance

The idea of "performing light" dates quite back in time, though it's relatively young compared to the performance of sound in order to produce music. The earliest known device for performing so-called "visual music" was built in 1734 by a Jesuit priest and mathematician, Father Louis-Bertrand Castel (Levin, 2000). Later examples of such devices appeared during the early twentieth century, like the Clavilux (Thomas Wilfred, first built in 1919, he also coined the name "Lumia" for this kind of "visual music") and the Lumigraph (Oskar Fischinger, patented in 1955, also appeared in sci-fi movie "The Time Travellers"), the latter being also intended to be performed along with audible music.

In this chapter we'll briefly describe some of the works about visual interfaces to both control and integrate music performance in some sense. While they're not actual examples of Lumia instruments – which is not what we're really interested in as far as this thesis is concerned – they still control music production through visual interfaces in many interesting, even entertaining, ways. Then we'll see how the connection between visual and audible performance can improve exploration and learning of abstract concepts of music theory. Finally we'll describe the Reactable, the platform on which the original work here presented, and we'll examine the motivation that lead to the development of this thesis.

1.1.1 Music Mouse

Music Mouse is a software developed by Laurie Spiegel in 1985. It is intended to turn the computer into a fully fledged musical instrument that can be used in live situations. For this reason, it doesn't allow the user to store anything for later playback.

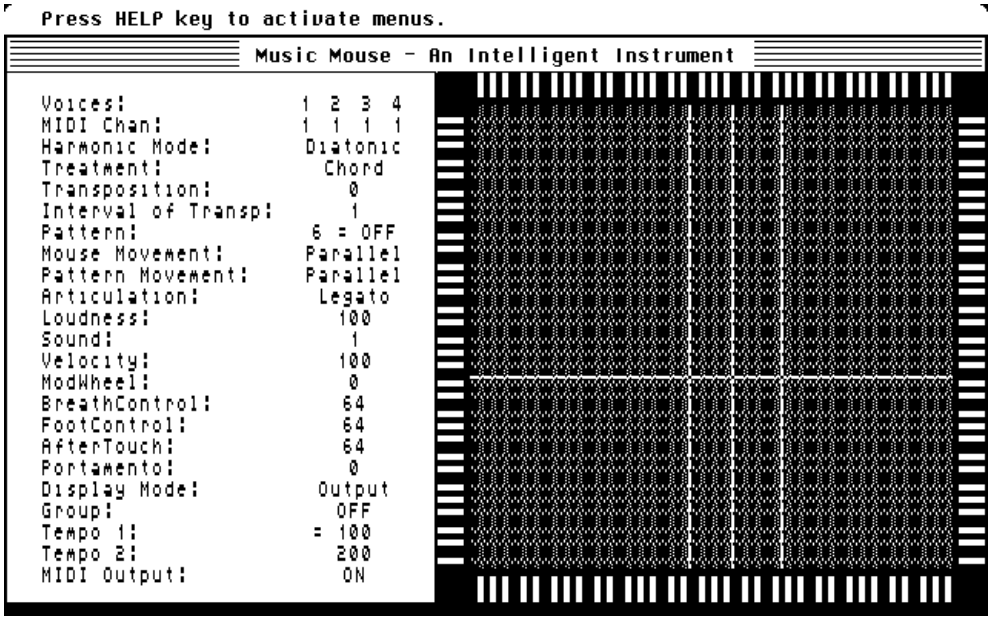


Figure 1.1. Music Mouse

Music Mouse uses the mouse as a controller (hence the name) and turns its movements inside a grid into harmony and melody patterns, thus requiring virtually no music knowledge to the performer, who can then completely focus on the direction he or she wanted the performance to take.

The software doesn't produce any sound on its own, instead its output is made of MIDI signals that can be redirected to external expanders or the internal synthesizer of the Macintosh OS – the platform on which it was first developed[1].

1.1.2 Instant Music

Instant Music is a software developed by Electronic Arts in 1986. Explicitly aimed at musicians and music lovers, it's intended to both assist them in creating original music, and support their performance on other instruments, thus giving an individual control and support of a full electronic orchestra.

The software interface gives the musician freedom to make variations on songs with no apparent limit, while a process of correction is actually working in the background to ensure that no "wrong notes" could ever be played. While this may seem a limit, it's actually a point of strength, since it allows inexperienced users to perform music without having to pay attention to details such as scales and harmony, while allowing experi-

1. Even if it's been ported to various other platforms, the sole implementation available on a modern and largely available platform is version 2.1.2 for Mac OS 9. Nonetheless the software seems to be still available for purchase through Spiegel's web site: `http://retiary.org/ls/programs.html`.

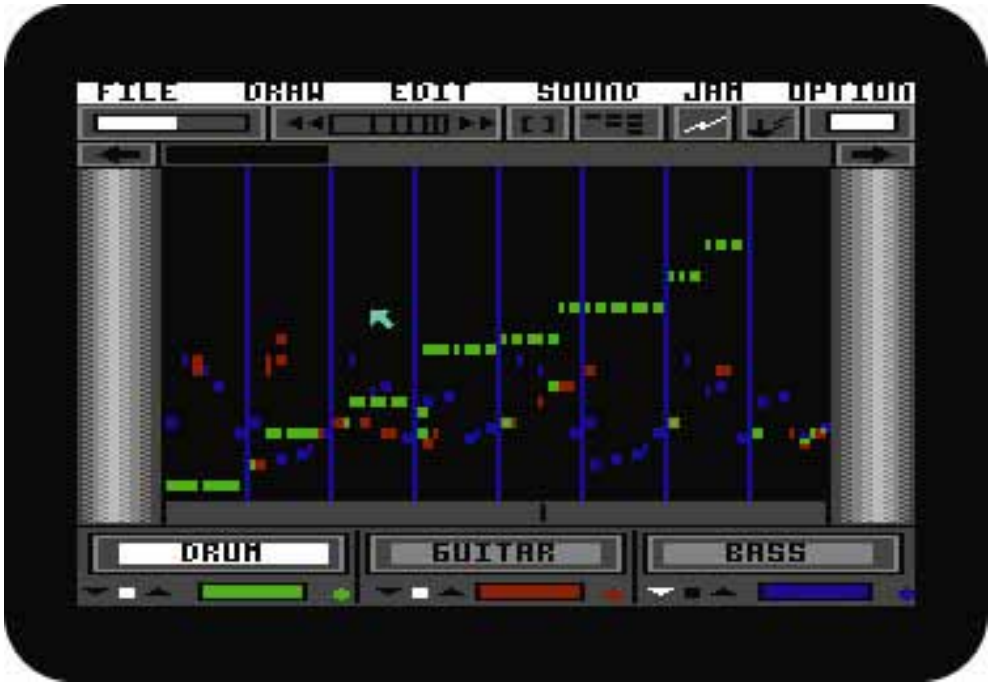


Figure 1.2. Instant Music

enced musicians to overcome this behaviour by using the software as a support while they freely perform on other instruments.

1.1.3 Auto pilot

Arguably the most interesting concept expressed by Music Mouse and Instant Music is to not let the user make mistakes, where a mistake means playing notes that don't fit with the tonality at any given moment.

Also, the idea of freeing the user of the burden of learning one particular instrument – that also means one particular interface to music – through an "agnostic" and easily understandable interface makes it easier for unexperienced users to interactively experience various, mostly subtle, concepts of music theory.

In fact there is some evidence (Seger, 1994) that individuals can learn a big deal of music theory – as well as other things – by simply being passively exposed to it, though this learning process is not explicitly supported by a formal description of the concepts. This process is called implicit learning and we'll discuss it in section 1.5.

Figure 1.3. Jam-O-Drum

1.2 Tabletop and tangible

Tabletop, tangible, touch-sensitive and other similar multimodal interfaces are not new concepts, as they can be found in early research projects, like "Urban Planning Workbench" in 1999, or even in popular science-fiction like touch-screen LCARS in "Star Trek: The Next Generation", 1987, or the tangible interface of the Asgard's computer core in "Stargate SG-1", 1994.

In recent years, some effort has been put in using multimodal interfaces for gaming and music purposes, as we'll see in the following sections.

1.2.1 Jam-O-Drum

The Jam-O-Drum (Blaine and Perkis, 2000) is mainly, but not exclusively, a gaming platform for up to four players[1], developed in 1998 by a team supervised by Tina Blaine at Interval Research in Palo Alto, CA. In its most recent version, each player controls a turntable-like interface with a big push button in the middle. Graphics are projected on the surface from above and sound is played through speakers in the room.

A number of different games had been developed, like "Bounce About", a Pong clone for four players, or "Learn Chinese" in which players cooperate to form Chinese ide-

1. The first version of the Jam-O-Drum allowed up to six or twelve players to play together.

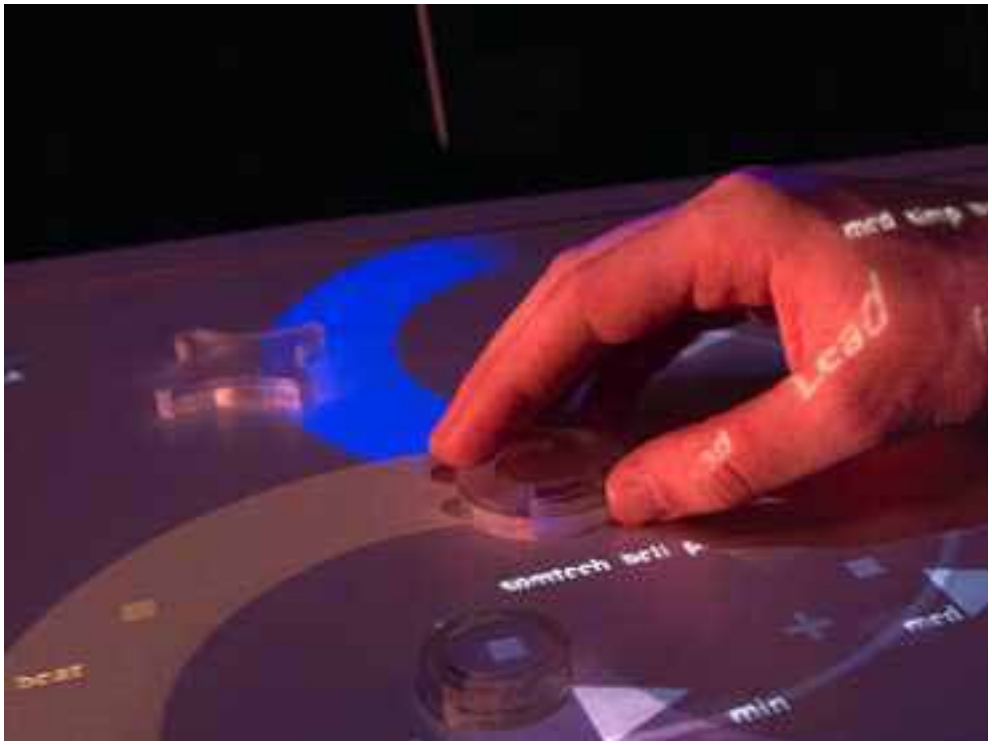


Figure 1.4. Audiopad

ograms that will eventually form a Chinese proverb. Also, a number of audio applications had been developed but these are usually collaborative games that produce music as a side effect.

The main focus of the Jam-O-Drum is on games, either collaborative or competitive, so the most important concept it features is that of multiple users working on the same device. As we'll see, this is a very important concept when we come to tabletop interaction, as tables are typical examples around which people aggregate.

1.2.2 Audiopad

The Audiopad (Patten et al., 2002) is an interface for musical performance that combines the modularity of knob-based controls with the expressive possibilities of multidimensional interfaces. Audiopad uses electromagnetic pucks (RFID) as objects associated with different functions, thus effectively turning a horizontal projection surface into a musical instrument that replicates the functions of modular synthesizers and samplers.

The Audiopad is slightly more flexible than the Jam-O-Drum, since it can do pretty much everything the Jam-O-Drum does, plus it's a complete electronic musical instrument. As we'll see in the next section, all the concepts we've seen so far have been put

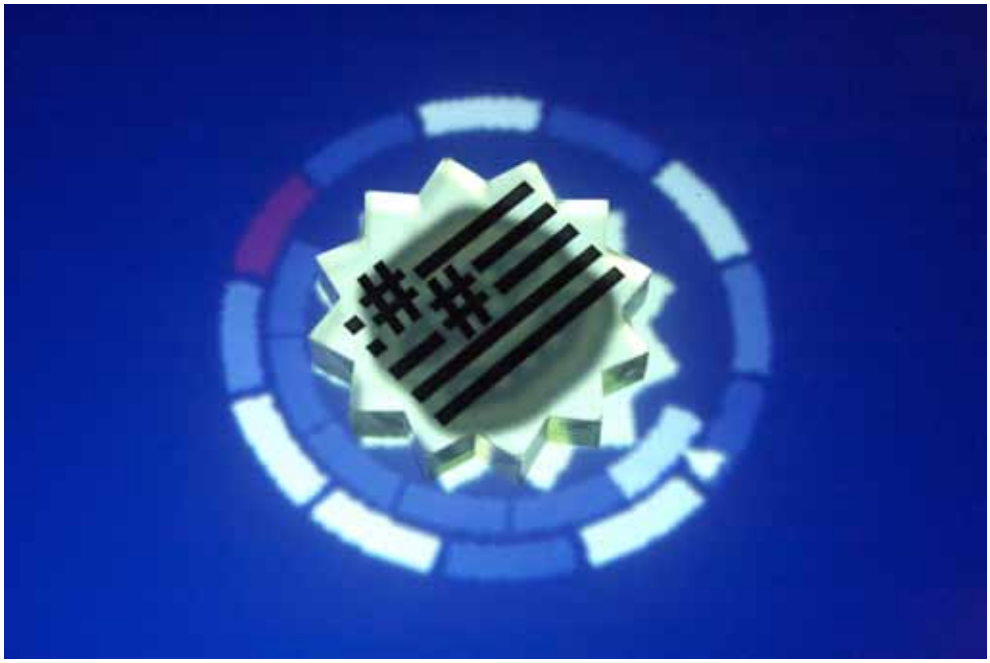


Figure 1.5. Current implementation of Tonalizer in the Reactable. Here we see the object featuring an outer ring in which users select the sole pitch classes they want to be playable, and an inner half ring in which presets for different configurations of the outer rings are stored.

together into the Reactable to some degree, thus making it a complete electronic musical instrument.

1.3 The Reactable

The Reactable is an electronic musical instrument with a tangible tabletop multiuser interface (Jordà et al., 2007) that's been developed within the Music Technology Group at Universitat Pompeu Fabra in Barcelona, Spain. It's designed as an instrument to make users explore and create soundscapes in the least possible intimidating way, thus allowing the widest possible range of users to successfully experiment with sound from the first moment, no matter what age or musical education level they are. This goal is achieved by presenting a user interface that adds sight to hearing and touch, thus combining audible and visual feedback to help users better understand and control what's happening at any given moment. Arguably, addition of visual feedback is the key point that makes the Reactable sensibly easier to approach than any other instrument, whose in turn don't usually present sound production and control in any different way than by hearing (see paragraph 1.5.4).

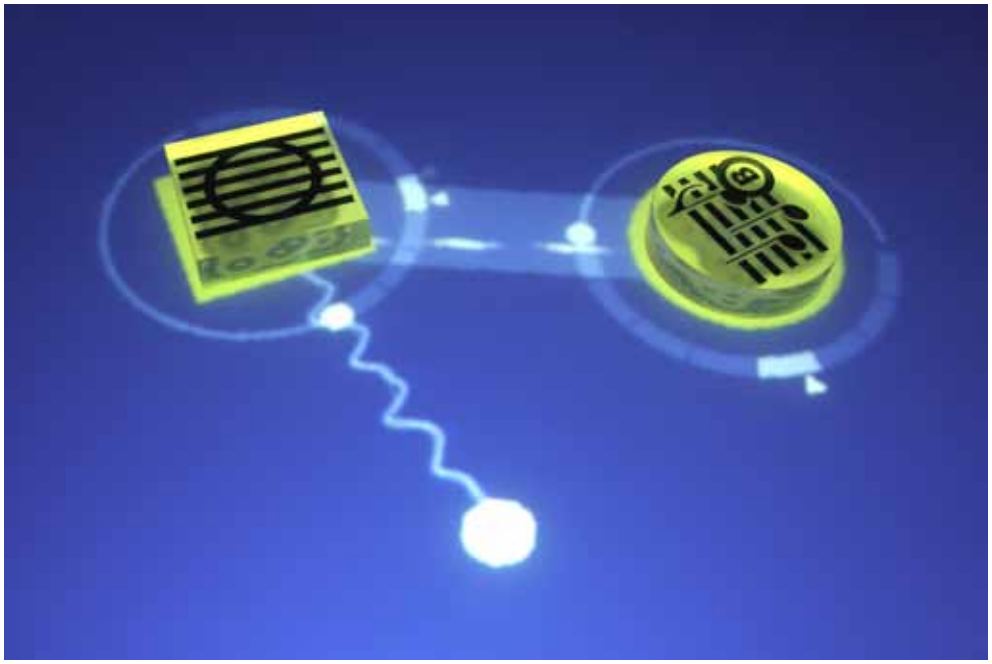


Figure 1.6. Current implementation of Sequencer in the Reactable. Here we see the Sequencer sending notes to a "plucked strings synthesizer". Around the Sequencer we see an upper half ring for volume control and a lower half ring for melodies that must be stored in advance.

The Reactable is a round translucent table that appears as a backlit display with which users directly interact using some special objects and their finger tips. Under the surface it features a fairly complex system made of infrared lamps, mirrors, an infrared-sensitive high speed camera and a common VGA beamer. Infrared lamps illuminate the bottom of the translucent table surface, thus allowing "inactive" areas to be ignored while "active" ones – such as fingers and tangibles – reflect the light and are captured by the camera. The captured video stream is then processed by a computer using reacTIVision, which extracts descriptors of the various objects it can recognize, and stream them over the network[1] to both the audio and visual synthesizers.

Both the synthesizers use the data sent by reacTIVision to produce audible and visual feedbacks. While sounds are simply played through a sound card, visual output is projected on the surface using the VGA beamer that's inside the table.

1.3.1 Fingers and tangibles

Most of the large multi-touch tables and screens nowadays available employ an interaction paradigm based on finger gestures. Users can manipulate visual objects on the screen

1. TUIO (Tangible User Interface Objects) encapsulated in Open Sound Control and streamed over UDP.

by "touching" them, dragging them around, enlarging or reducing their size and much more. The Reactable sticks to this paradigm while extending it through the introduction of objects that can be physically grasped by hand, put on the table, moved around and eventually removed when they're no longer necessary. These so-called "tangibles" are identified and tracked by reacTIVision, the open source computer vision engine that sits between users and Reactable's proprietary audio and video synthesizers. This software identifies and tracks special fiduciary markers attached to the tangibles, so that various parameters – such as position, orientation, linear and angular speed – can be extracted. Each distinct fiduciary marker is associated with a different function, for example sound generators (i.e. noise, waveforms, physical modelling), filters (i.e. low/high/band-pass, resonating), modulators (i.e. LFO), audio samples, etc. These building blocks make the Reactable an almost-full featured "analog synthesizer". Some of these parameters are linked to position and rotation of the associated tangible, while others are controlled via finger interaction with a minimal GUI that's presented around the object, in some sort of "augmented reality" fashion. For example, a sine wave generator controls frequency by rotation and amplitude via a gauge put around it.

The strength of this approach is that users have a real grip on what's going on with the production of sound, thus making the Reactable a more realistic musical instrument, while freeing the user from the responsibility of continuously controlling sound emission, since it's automatically generated, thus allowing him or her to concentrate on performance direction.

For a full description of other objects and functions, see Jordà, Geiger, Alonso and Kaltenbrunner (2007).

1.4 Why music theory on the Reactable is desirable

The Reactable has been originally thought and developed as an instrument for free exploration and production of soundscapes, a way for experimenting with electronic music without being bound to any music heritage, therefore it allows manipulation of every sound parameter, be it pitch, volume, timing and so on.

During its development, it started to gain international fame, mostly over the Internet thanks to some videos that explained the basic functions of the instrument. Musicians started to experiment with it and some of them could even bring it to their stage performances. In 2006, icelandic singer-songwriter Björk decided to have one in her "Volta" tour, even if it hasn't been actually used to its full potential, but for playing prerecorded loops with a high choreographic value.

"Traditional" musicians started to feel that all this freedom of control over parameters was more a limit than an opportunity, after all even the early analog synthesizers had a

way to play pitch classes with a keyboard. In response to this, a set of objects was developed. It consisted of a very simple MIDI sequencer (that played preloaded melodies) and an object, called Tonalizer, that was intended to limit what frequencies the tone generators were able to play, thus effectively binding them to pitch classes of western tonal music. Since the original team didn't feel this was a feature they wanted, they decided to develop it at the minimum level of usability and then leave it as it was.

The work here presented has to date a lot of unexpressed potential that shall be properly explored and evaluated as development continues. It can turn the Reactable into a complex yet easy tool for improvisation and composition, as much as Music Mouse did with personal computers; it can also turn it into a composition tool that can support performance of other musicians[1], given an appropriate interface with musicians, like Instant Music, or even with other similar instruments. It can also turn the Reactable into a learning aid for music students, as we'll see in the next section.

1.5 Implicit learning

Implicit learning is the process through which an individual becomes sensitive to the underlaying regularities of highly structured systems, like language or music. Even if such knowledge remains at a level such that one is not able to explicitly describe them, it influences perception and interaction with the environment (Seger, 1994). The most prominent real life example of implicit learning is of course natural language. Babies learn to speak at an early age, first by merely imitating sounds produced by other people, then by learning how concatenation of such sounds conveys a message through sentences. But the first time they explicitly understand how those sentences are composed and why different sentences mean different things is at school, where they're formally taught the rules of language through reading and writing.

For an in-depth discussion on implicit learning, see Bigand, Lalitte, and Tillmann (2008) and Seger (1994).

1.5.1 Music as a natural language

It's been argued that the origin of music itself may be similar to that of natural languages (Molino, 2000). Considering music as a natural language would then mean that it has its own set of symbols, words and sentences, all tied together by a grammar, that is the set of rules of a given musical system. In this sense, each musical system is a different natural language as much as English and Italian are different natural languages. Also, generative grammar approaches have been used in musicology for analysis of musical

1. e.g. tonalizer based on scale modality, automatic progression generation, algorithmic solo improvisation…

pieces (Ruwert and Everist, 1987) though the idea of a "universal grammar"[1] has not received much consensus, while the evidence of specialized grammars per author or per period is much more accepted.

Indulging in the hypothesis of music as a natural language, we can also assume the existence of a transformational grammar that allows people not only to understand new meaningful sentences and reject those that don't make sense, but also to produce new, possibly never heard before, meaningful sentences. For more information on generative and transformational grammars, and their relationship with origin and understanding of language, see (Chomsky, 1965).

Assuming that this hypothesis holds, it is then reasonable to suppose that all the concepts already developed for natural languages may hold for music too, thus the following discussion will make sense.

1.5.2 Music education

Music is traditionally taught by teaching how to play a particular instrument – that is the experience of sound – while also using it to teach the underlaying rules in a formal way. This approach allows students to learn music theory by direct experience on a chosen musical instrument. On the other hand, this approach forces students to learn two nontrivial matters at the same time, possibly inducing a bias towards the chosen instrument, given the fact that not all instruments have the same possibilities – e.g. piano is a complete harmonic and melodic instrument, while trumpet is just a melodic instrument[2]; also some special rules developed for church organ are quite different from those developed for piano.

To reduce the complexity given by learning two complex matters at the same time, it seems reasonable to separate these two tasks, and possibly perform them in sequence. While it is not feasible to proficiently learn how to play an instrument without learning even basic rules of music theory, it may be the other way. A student can learn a lot of abstract concepts that can then be applied to any specific instrument at the sole cost of actually learning to play it – that is learning the technology behind it. This may – or may not, it should be properly evaluated – reduce the time required to learn how to play an instrument, similarly to how already knowing an instrument usually reduces time to learn an additional instrument.

---

1. The term "universal" most reasonably refers to all music compositions under a single harmony system instead of a grammar that describes all the possible harmony systems, the latter being a fascinating hypothesis, though still unproven.

2. This, among other reasons, is why piano is often taught as a complementary instrument. We are not implying that learning piano is a bad thing on its own, it just adds complexity to the study of the primary instrument.

1.5.3 An easy, non-intimidating, musical instrument

What we lack now is a way to effectively practice those abstract concepts that music theory is made of[3]. As we've seen, we can assume that most of these concepts, even subtle ones, have already been implicitly learnt by students. What we actually need is a way to explicitly express them without forcing students to master a traditional musical instrument, which can be either a delightful or excruciatingly painful experience, depending on the student's attitude towards the instrument and its study. As we've seen in section 1.3, the Reactable is an intuitive, easy to play, and – most important – non-intimidating musical instrument by design.

As it's detailed in (Bigand et al., 2008), regularities and relations between tones, scales, chords, etc, are important when it comes to implicit learning, and learning in general. This is the key point they use to argue that learning western tonal music can be optimised and improved using multimedia tools that emphasize such structures.

The work presented in this thesis aims at giving the Reactable a way to perform with music theory, thus turning it into a musical instrument in a more traditional sense, by integrating notions of musical structures and presenting them as an optional operating mode. The intuitive and easily graspable interface of the Reactable makes it a perfect candidate to give students the ability to experience musical concepts by reducing the complexity of learning a traditional musical instrument, while leveraging on the implicit experience of music theory one may have unconsciously acquired. However it is clear that this thesis builds upon the existing Reactable so at any moment it's possible to simply ignore all the music theory framework and revert to the original "agnostic" behaviour of the instrument, if desired.

It's also worth noting that even if Bigand et al. only analyse western music theory, it can be argued that regularities and relations between other cultures' notions of musical structures exist, though they can be quite different from those existing in western music. However, if a tool is properly designed to be flexible and extendable enough, it should also be easy to adapt it to different rules, and this is one of the main goals that drove the design process we are going to detail in Chapter 2.

1.5.4 Visual feedback

Arguably the key point in making the Reactable so immediate to approach that even infants can play with it is the visual feedback users are given about the running processes at any given moment. In fact, since visual exposure is much more constant and thorough during life than music exposure is, shape and visual pattern recognition be-

3. There is much empirical evidence that exercise teaches much more than simply watching or listening someone else exercising. Even if abstract concepts can be learnt at a subconscious level by simple exposure, it seems reasonable that explicit formalization of concepts may be easier by practically experiencing them on an instrument.

comes a granted, automatic and quite sophisticated skill – for non-blind people. Also, coordination between sight and action is usually more natural and sophisticated to most people, where coordination between sound and gesture is a process that evolution put at quite a low level, mostly connected with reaction to immediate dangers, survival and functional communication. In this sense, the Reactable makes of visual feedback a point of strength when it comes to controlling sound and music.

On the other hand, visual feedback has a subtle drawback, that is information overload. A badly designed interface can present too little or too much information. While presenting too little information usually does no harm other than limiting users' knowledge about sound production processes – which is actually what traditional musical instruments do – giving too much information may induce confusion in users and clutter the visual interface at a point that it's not usable anymore.

A lot of care has been put into good design in the Reactable, and the work here presented tries to adhere to those principles by presenting the most minimal yet expressive and intuitive interface possible to easily explore the widest possible range of the system's capabilities.

2

Design process

This chapter documents the phases of the project. First, we'll review the requirements of the project, then we'll go through the design process by seeing every idea in order of appearance, we'll point out the reasons for every choice, we'll detail both strong and weak points of every idea and see why some of them made it to the final concept while some other died trying.

2.1 Requirements

The basic, most important requirement that sits behind all this work is that of having a way to bring western tonal harmony on the Reactable, and making it possible to control the way other sound-generating objects work, for example by restricting the frequencies of a sine-wave generator from a "continuous" spectrum to a set of pitch-classes. Although not originally required, the system has been designed to not be restricted to western tonal music, but to support a generic tonal system, as we'll note in subsection 2.4.1.

The second requirement is to present the user with an interface to modify the behaviour of the Reactable in order to fit a given tonal system and allow to explore, combine and take advantage of the harmony's rules in order to support or even create a performance.

Last but not least, the resulting system should present a minimalist and intuitive yet complete and powerful interface to achieve its goals. This is most important when thinking about Reactable's target users, since its main design goal is to be an easy, non-intimidating musical instrument that anyone can approach and start achieving appreciable results in very little time and with virtually no musical knowledge. Also it's important when thinking about aid to music education, as discussed in the previous chapter.

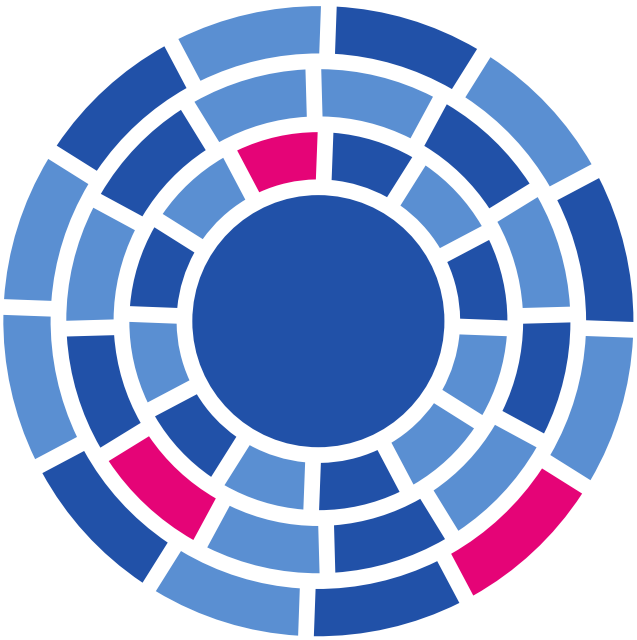


Figure 2.1. circular accordion

2.2 The circular accordion

The very first idea was the object shown in figure 2.1. The circle in the middle holds place for the tangible object and the rings around it are the actual interface. It is a very compact design that integrates both the tonalizer and the melodic sequencer within a single object. As we shall see, less is not always more, and all this compactness eventually proved to be quite a poor solution with a few very big problems.

2.2.1 How it works

The basic idea behind the rings is that each ring is a step in a sequence advanced by an external step sequencer. This sequence represents either a chord progression or a melody. The innermost ring is the first one in the sequence while the outermost is the last one, and there can be arbitrarily many rings, thus allowing to build sequences of different lengths. The rings are divided into twelve sectors that represent the twelve half-tones of the chromatic scale[1] and they can be rotated in order to align different keys that can make up either a progression or a melody, thus allowing to select such a sequence with a gesture as simple as a straight stroke from inside to outside. In figure 2.1, if we see the red

1. Here we use the western chromatic scale as a reference but it's worth noting that this design is also suitable for the Shi Er Lü (Chinese chromatic scale) or the Sagram (Indian solfege) with no modifications in the interface (see §2.4.1).

Figure 2.2. second iteration of the tonalizer (play mode).

keys as C, and the light and dark keys as the white and black keys on a piano keyboard, we can easily see that the keys are aligned with a major third interval between the first and the second ring, and with a minor third interval between the second and the third ring. If we draw a stroke starting from a key of the innermost ring and going along an ideal radius of the circle, we clearly see that we can make either a I-III-V progression, if we are working with chords, or the corresponding arpeggio, if we are working with notes.

2.2.2 How it fails

The big issue with this design is that it really tries to solve the whole problem with a single object. Although this may seem an elegant and compact solution, sometimes less is not more. Let's see how this design is not a viable solution.

1. As the sequences become longer, the interface tends to grow very quickly in size, and take up most of the space on the table. Although this space is not physical and can be redeemed by hiding the interface when it's not being used – not to mention that it can even be cleverly reduced during editing – working with sequences of as little as six or eight steps may still become inconvenient. In fact, at any given time, the visual interface may intersect with other objects, both tangible and visual.
2. When editing a chord progression, there's no obvious way to select a particular mode for a given ring (e.g. major, minor, seventh…) as well as to choose a scale that fits a given tonality.

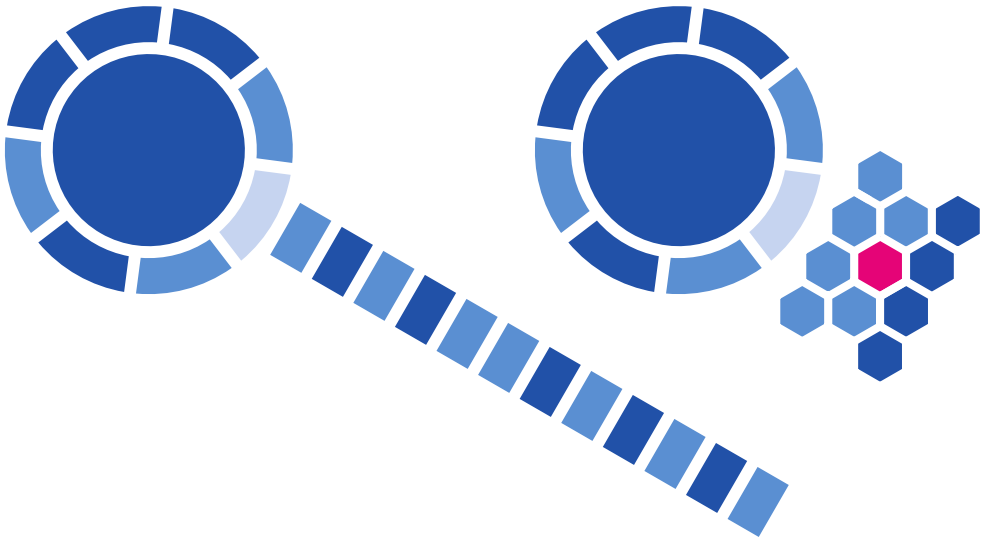


Figure 2.3. second iteration of the tonalizer (slot editing mode).

3. It relies almost completely on an external timing device, namely a step sequencer. While this is not a bad thing at all, we'll see later that there are a number of things that can be done with internal timing, using an external step sequencer just as a timing reference.

2.3 Smaller bits

The second iteration proves that, sometimes, less is actually more. Notably, this time the proposal involves splitting the "circular accordion" in two parts: one dealing with chords, and the other dealing with melodies. This way, while it actually has "more" objects, we have them solving "less" – and smaller – problems each. Not only this makes things easier to deal with, but also allows for more sophisticated control associated with each object.

2.3.1 Tonalizer

The second iteration of the tonalizer is shown in figure 2.2. The first notable thing is that it still carries one ring of keys around the tangible object. The second thing is that it only carries one ring of keys around the tangible, and it also has less keys than the accordion. There is nothing wrong with having twelve keys around the object, but this time the keys are not related to notes in scales: in fact, they are slots in a sequence, and each slot accounts for a chord. This means that we can have eight as well as four or sixteen slots around the object, and this amount limits the maximum length of a sequence at a given

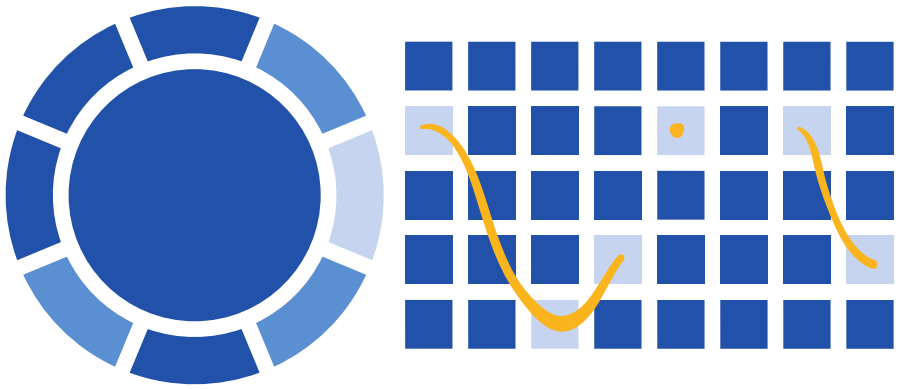


Figure 2.4. second iteration of the melodic sequencer (editing mode).

time[1]. Another notable thing is that the slots can either be active or not. In figure 2.2 the inactive slots are the darker ones, the active slots are those slightly lighter and the single brightest slot is the one that's currently being played. In this iteration of the tonalizer, a sequence can be of any desired length, in fact, even if practical limits still exist, it is a lot longer than before. Moreover, this arrangement also allows for more intuitive reordering of the sequence by dragging slots in the desired positions, and only active slots are actually used in the sequence, while inactive ones are simply skipped. This allows for arbitrarily long sequences up to the number of available slots around the tangible.

Figure 2.3 shows the tonalizer while editing a given slot: in this case it is shown editing the active slot, but any other slot can be edited, and even more than one may be edited at the same moment. Changing the chord of a particular slot influences behaviour of those objects that rely on it (e.g. the melodic sequencer) whenever that slot becomes active. As we can see, editing of a slot can be performed with a number of different interfaces: for example, on the left side we see a piano-like keyboard on which the user simply chooses the notes and makes up the desired chord, while on the right side we see a harmonic table arrangement that still represents the same amount of notes of the piano-like keyboard while providing a far more compact design.

2.3.2 Melodic sequencer

The other object resulting from the split of the circular accordion is the melodic sequencer presented in figure 2.4. When it's operating in play mode, it looks just like the tonalizer described above, and the ring of keys behaves pretty much in the same way. The only difference is when the object is put in edit mode. The grid interface shown in figure 2.4 represents a "measure", here divided in eight beats of equal duration, and five

1. While theoretically there's no limit to the amount of slots around the tonalizer, with this representation there's a practical limit given by the resolution of the beamer, the size of the tangible, and the minimum size of a blob to be effectively recognised as a finger.

rows that represent the notes that the user can choose to compose melodies.

The most interesting part of this design is the way it exploits information given by the tonalizer to give users a safe way to compose "correct" melodies, where "wrong" melodies are those made of notes that "sound wrong" with a given chord. The key point is essentially based on the fact that a given chord imposes a tonality and there are scales in that tonality that don't contain notes that would be inharmonious when played together with that chord. For example, if we're given a major scale and we play the 7th or diminished 5th along with the corresponding major chord, we create a tension. While this is a legitimate artistic choice when consciously made by a skilled composer that knows exactly how to deal with and resolve such tensions, it is also something that comes unexpected to the listener – and that's what makes music interesting – but it's also perceived as an error when improperly used. The naïve solution is to not permit the casual user to make such "mistakes" while still giving the skilled performer the chance to play everything it's felt appropriate at any given moment. How exactly the system decides what scales fit a given chord is an implementation detail (see Chapter 3).

The grid featured in figure 2.4 is a measure divided in eight beats and five notes: this means that it has a resolution of ⅛th and a pentatonic scale (either major or minor, depending on the tonality). The way users can compose melodies is by drawing strokes and taps with their fingers. The strokes are then analysed and some predefined significant points[1] are used for triggering on and off the notes on the grid. This results in an extremely playful yet powerful interface to a rather complex task like music composition.

Regarding the relationship between melody and chord slots, there can be at least two levels of binding, here discussed by increasing complexity.

1. The sequences are strictly coupled: both objects have the same maximum amount of slots, and both sequences have the same length. Activating or deactivating a slot triggers the same action on its corresponding slot in the other sequence, and if a sequence is reordered, so is the other.
2. The sequences are loosely coupled: each object can have a different maximum amount of slots, and each sequence can be of any length. Both the sequences are advanced as usual and each time one sequence reaches its end, it is looped. Due to the length difference, this behaviour can lead to both interesting and unexpected scenarios in which a melody is played over a different chord than the one over which it was composed, and possibly even a different scale.

It's worth noting that there's still the possibility to put on hold the advancement of one or both the sequences, thus either making the same melody play with different chords, or making the same chord serve as a basis for different melodies. This, combined with

1. For example taps, or "zero-derivative" points where the derivative is relative to the time axis of the grid, or points where the finger slowed down for a while, etc…

the latter scenario, opens for an interesting challenge. Each time a chord is modified, or even a different melody/chord pair is put together, the set of feasible scales is possibly different. While some scales currently in the set associated with the old chord can still fit the new chord, others may not, and even new ones may come into play. If a melody is composed on a certain scale and the new chord doesn't fit that scale anymore, we may take two actions: completely discard the melody, or try to adapt it to one of the new scales that fit the new chord. The first action is the most simple to implement but it has the disadvantage of taking a possibly unexpected choice with which the user may disagree. The second choice opens for a very interesting challenge, that is how to translate the melody to fit one of the new scales. Among the possible solutions, these two may be the most interesting yet difficult to pursue.

1. The raw user's input (i.e. the strokes and taps) may be kept in order to "replay" the gestures over the new scale. Now the problem is whether to adapt the gesture to the new size of the interface, or to leave it unchanged and just replay it over the interface – thus also having to decide how to align the gesture with the new interface and what to do if it exceeds the bounding box.
2. The melody may be transposed to the new scale using a set of rules that tries to minimize the difference between the old and the new melody. This set of rules should come from the underlying music theory.

Both cases may either ask the performer to choose the target scale or automatically select one among the "most similar" ones to the original scale. While both these solutions are probably the most "correct" regarding user expectation, they are quite complex and difficult to implement. Given the small amount of time the first naïve approach of throwing the melody away has actually been adopted for the demo application, while the whole "adaptive melody" concept will be considered as a future development.

2.3.3 Disadvantages

The major disadvantage with this iteration of the tonalizer is that it still requires some knowledge of music theory. In fact, the chords are specified via single notes on some kind of keyboard interface. The only real improvement over the existing tonalizer is the one related with inferring and using tonality and scales to give hints to the relevant objects, which is interesting indeed, but unfortunately it's not very much.

The other disadvantage of this second iteration is that both the objects may rely too much on an external timing device. Although this is of arguable use with the tonalizer, the melodic sequencer has no way to express different durations other than by relying on a properly designed step sequencer. More generally, a real step sequencer is not really useful on the Reactable, while a timing reference object (like a metronome) is definitely more useful – not to mention that it already exists.

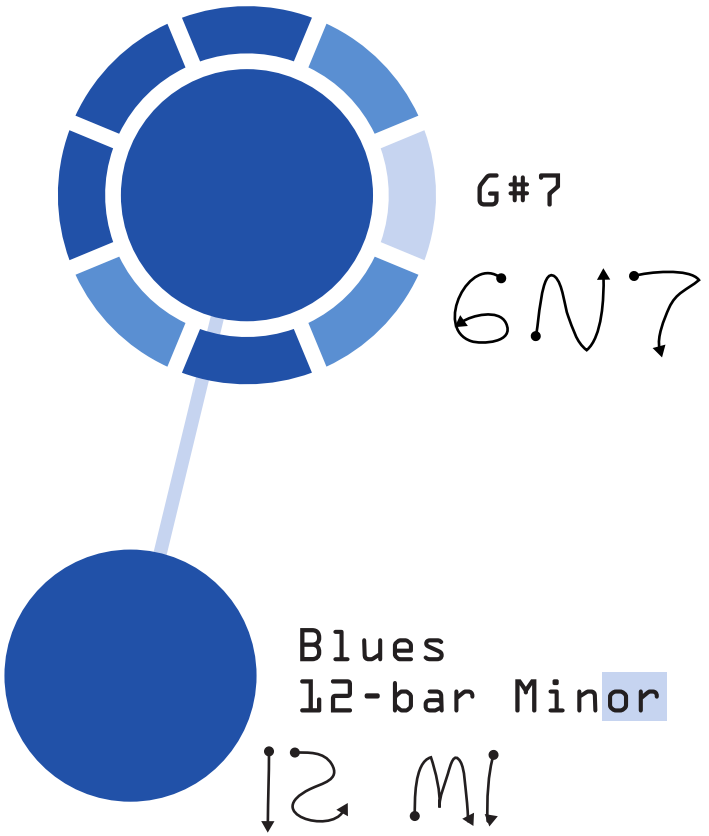


Figure 2.5. third iteration of the tonalizer (editing mode).

2.4 A step forward in user-friendliness

The third iteration tries to solve both the major disadvantages we've just seen. Most of the work has been done on the tonalizer, introducing a completely different input paradigm, which exploits the tactile interface to give the user a more natural, and even more abstract, way of choosing chords. On the other hand, the melodic sequencer enjoys a minor yet important improvement which greatly increases its flexibility while also removing the need for an overly sophisticated step sequencer.

2.4.1 Tonalizer

The problem with requiring some knowledge of music theory is not with the level of knowledge required, but with the requirement itself. If we want the tonalizer to be an effective tool for the unexperienced user to experiment, enjoy, and eventually learn some concepts of music theory, the level of knowledge required should ideally be zero. The tactile input paradigm is an extremely powerful yet immediate mean of interaction and, as we'll see, the level of knowledge that we will require is way more basic than the most

basic level of music knowledge.

The third iteration of the tonalizer shown in figure 2.5 features handwriting recognition. Given the example depicted in figure 2.5, it may seem that some strong assumptions are taken: Latin alphabet, Arab digits, Anglo-Saxon conventions for chord notation, etc. In fact, this is not the case at all. The power of this approach is that it is flexible: any kind of symbols, like existing alphabets, music notations, or even artificially constructed sets of symbols can be implemented. For example, it may be interesting to implement abstract symbols, taking advantage of the natural ability of the brain to recognize and re-create shapes and figures, thus separating the task of experimenting with music theory from the task of learning and using specific notation systems that are usually tied to specific languages and cultures. This may also prove particularly effective when the instrument is explicitly used to teach the basics of music theory, since it relieves the student of the burden of a specific set of conventions.

Finally, it's worth saying a word about handwriting recognition methodology. Figure 2.5 features a single-stroke glyph-based variant, a concept quite similar to the one found in early Palm PDAs. The demo application actually features this kind of algorithm, based on modelling a single-stroke Bézier curve of arbitrarily high degree out of user's input, and matching it with a set of prebuilt models. This is probably the most naïve algorithm that still retains a sufficient degree of flexibility, since it has to be both position and rotation independent. The handwriting recognition system[1] is designed to be as flexible as possible, so any kind of sophisticated methods – from very simple approaches to highly advanced algorithms involving artificial intelligence or computer vision – can be implemented.

Figure 2.5 actually features two objects. The one at the top is the tonalizer as we already know it. It still has the ring of slots around it[2] and it also features handwriting recognition. The object at the bottom is a sort of cadence chooser. This object works as an auto-pilot for the user who doesn't want to put much effort in creating a chord progression, but wants to concentrate on other aspects of the performance. The core idea is that the user selects a single chord on the tonalizer and then chooses a "style" from the other object. Then the style object creates an entire progression taking the chosen chord as the base key. For example, figure 2.5 features a twelve-bar blues progression in G♯ – the 7th is discarded as it makes no sense[3]. Both the objects feature handwriting recognition using the single-stroke glyph-based approach mentioned above, but, while the tonalizer

1. As we'll see in Chapter 3, the entire demo application is based on plug-ins.

2. It doesn't appear obvious how a specific slot is edited. This issue is addressed in the final iteration, which is the prototype that is implemented in the demo application.

3. Probably a jazz turnaround like ♭III^{7}-ii-V-(I) would have made more sense with this figure, but it's not really a complete progression, thus making a bad example.

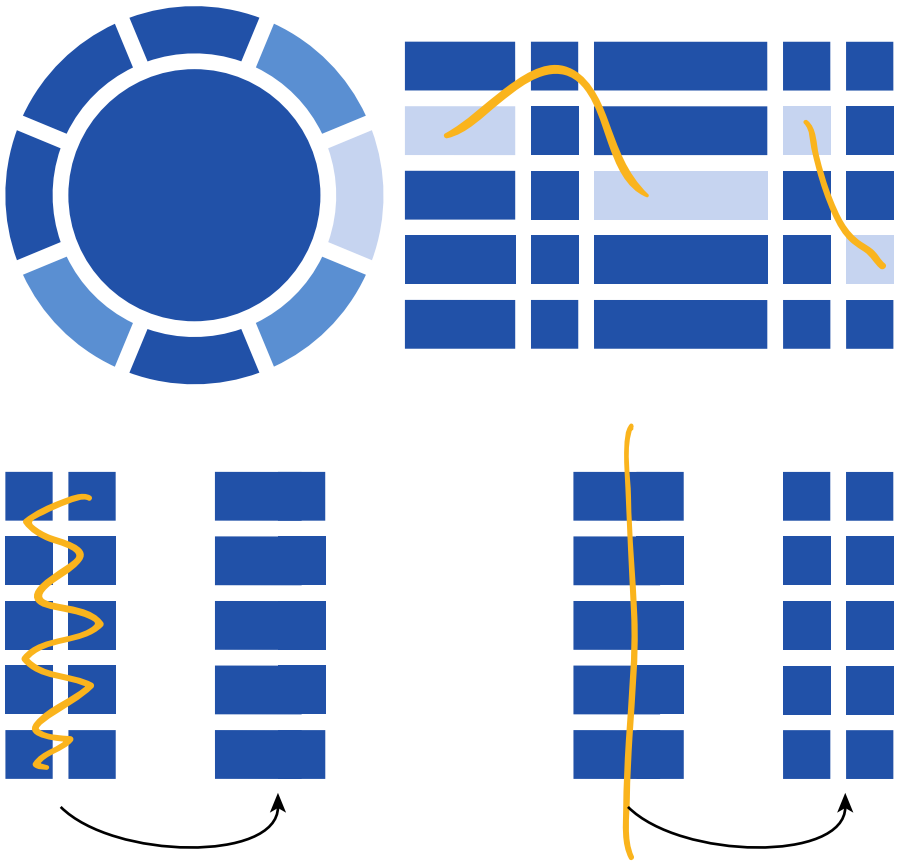


Figure 2.6. third iteration of the melodic sequencer (editing mode).

uses a direct constructive approach, the other object features a list oriented selection paradigm, in which the first possible choice is shown in some sort of auto-completion interface, and the others, when appropriate, can be chosen by expanding the list.

Another option has been considered, that is the inclusion of a "legacy mode" in which, for each chord in the sequence, the tonalizer reverts to the current interface that displays the ring with twelve keys and lets the user edit the chord note by note. Even if it may seem a great benefit for the musically skilled user, it still poses an ambiguity about how to interpret the chosen set of notes. For example, if the user selects a major triad, it is not clear whether the system should use just those three notes, like the original tonalizer does, or if it should detect a major chord and behave like the new version that's been designed so far. For this reason, the "legacy mode" has been discarded.

2.4.2 Melodic sequencer

As it is clear from figure 2.6, the melodic sequencer has not been subject to deep modifications. The only difference with the previous iteration is the possibility to specify

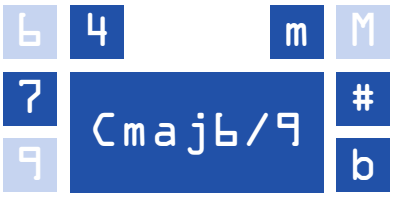


Figure 2.7. the chord widget, for direct editing of chords.

different durations for the columns of the grid. This completely removes the need for a step sequencer, allowing for a more general metronome-like device to be used as a beat reference, thus requiring the step sequencer to be implemented inside the melodic sequencer. The advantage is quite clear: using the simple gestures depicted in figure 2.6, the user is able to create melodies of increasing complexity. The two major drawbacks of this approach, even if they are in fact rather minor, are that

1. it may be possible to create unusual durations, like triplets, that the user may be uncomfortable with, and
2. there's no way to create polyphonic melodies with different durations within the same object.

The solution to the first drawback may be as simple as the creation of an invisible underlying grid on which to snap the split points. For the second drawback, either a user may put on the table more than one melodic sequencer associated with the same tonalizer, or a possibly quite complex three-dimensional interface can be developed. The former solution is arguably the most feasible one, as it doesn't require any additional effort to be implemented.

2.5 Final implementation

Most of the concepts so far discussed have been kept, while some had to be excluded by the final iteration or the demo application due to many reasons, most notably for not being practical or for requiring more time than available for implementation.

While the melodic sequencer hasn't had any modification since the third iteration, it features some notable differences in the demo implementation: it hasn't the possibility to create columns of different durations, and there's no explicit way to choose a specific scale from the list of those that fit the current chord. In fact, even if all the possible scales are actually computed, the first one in the list is presented to the user.

On the other hand, early tests revealed that the naïve algorithm for handwriting recognition wasn't always reliable. This is most probably due to the unrefined implementa-

tion of the demo application, possibly combined with the low resolution of the video camera. Moreover, time constraints prevented the development of a more sophisticated, and invisible, interface for chord manipulation. For all these reasons, the chord widget shown in figure 2.7 has been developed. It serves the two purposes of clearly displaying the chord being edited and of permitting direct, more precise, manipulation of the chord it represents.

It is interesting to note that, since there can be more than one such widget on the table at the same time, more than one user can edit the chords in the sequence, even if only one of them can edit chords via handwriting. This is because it is not clear how to associate a stroke to the chord that the user wants to manipulate. At this point, further research and evaluation are needed to clarify both the usefulness of having a separate chord widget, and how to associate strokes with chords.

3

Doodle: a proof of concept implementation

After the design phase described in Chapter 2, a demo application was developed in order to effectively evaluate and assess the project. The result is a project called Doodle which is logically divided into sub-projects.

Doodle: this is the main application that acts as a supervisor, makes communication between components possible and also produces visual feedback for projection on the table surface. Doodle is actually split into a SDK which is necessary for effective code reuse, and the actual application, as we shall see in sections 3.1 and 3.2.

Glyph: this is the gesture recogniser responsible for handwriting recognition, as explained in section 2.4.1. It features a plug-in architecture for handwriting recognition engines and it even masks internal Doodle details, like Cursors, in order to allow development of engines without depending on Doodle. In order to demonstrate the architecture, two plug-ins were developed, as we'll see in section 3.4.

Tonalizer and Sequencer: these are the applications that have been discussed in Chapter 2. We'll see implementation details in sections 3.5 and 3.6.

3.1 DoodleSDK

As it turned out, the Reactable implementation available at the time this work was developed was a mixture of various components written in many different languages, like Java, Pure Data, C++, and so on. Although it is not impossible to develop an add-on, it is just impractical to do so with little knowledge of the system and little time to work, so the best solution was to work from scratch, directly using reacTIVision and the TUIO C++ library. For this reason Doodle was designed from the ground up as a general C++ framework for tabletop applications with a tangible and multi-touch interface.

Doodle features a plug-in architecture that's used to develop applications – i.e. objects that gather information from the environment, communicate with each others,

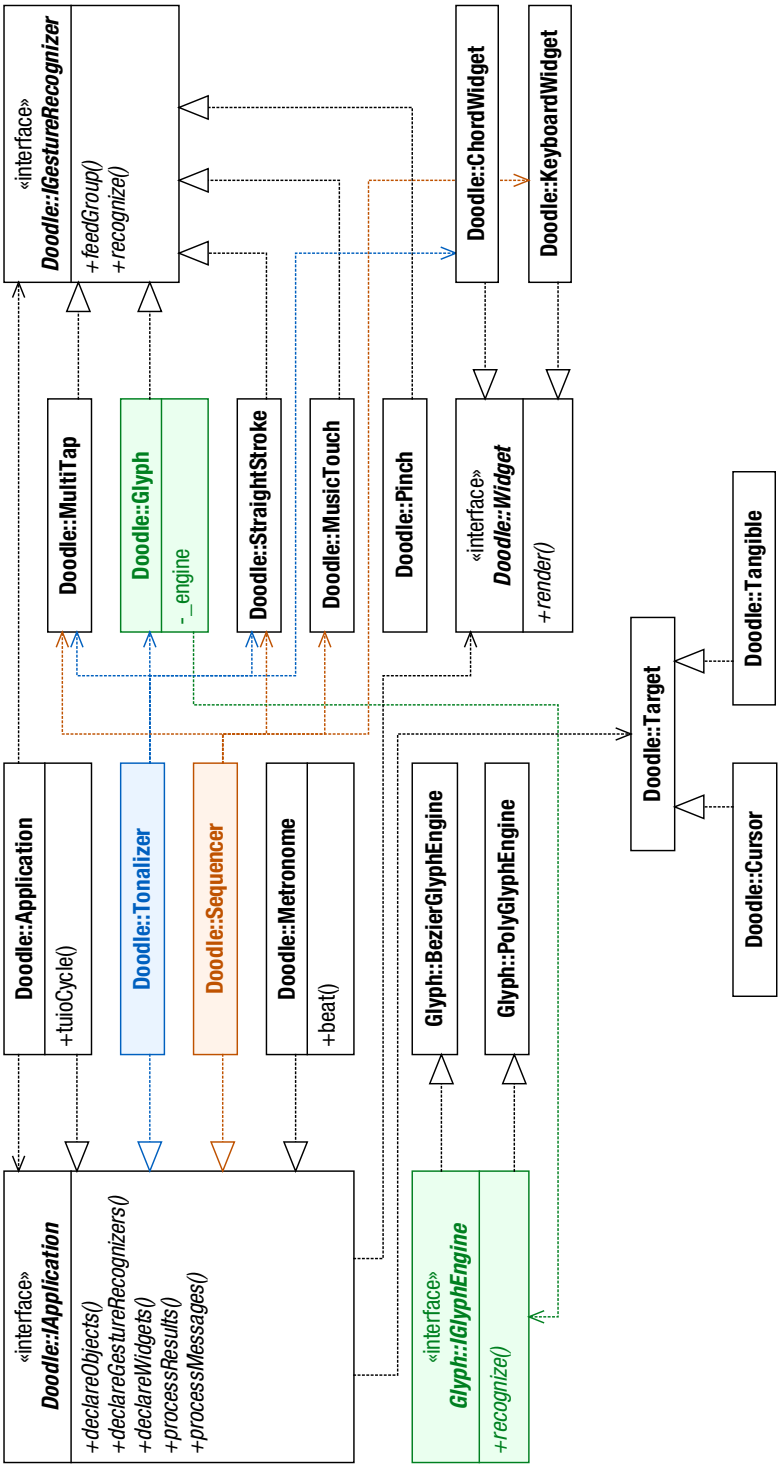


Figure 3.1. Simplified class diagram for project Doodle.
It's important to note that colours are used for better legibility, not for stereotyping.

and even produce audible output – and gesture recognisers – i.e. objects whose purpose is to analyse the gestures performed by users with their fingers and communicate the results to applications. The project was developed with Nokia Qt Framework 4.5. The positive aspect of this choice is that Qt Framework is a widespread, well known, cross-platform and STL-compatible C++ framework for desktop and embedded applications, and the key point that led to its choice is the simplicity of its plug-in infrastructure, as opposed to the wide range of different solutions for dynamic object linking on different platforms.

The downside of this choice is that Qt Framework 4.5 does not support typical multi-touch features like gesture recognition and multiple pointers, while latest stable release 4.6, though it was unavailable at the time, does. For this reason, such infrastructure had to be developed, and much care was put into design in order to make it as much adherent as possible to Qt's guidelines, so that a port from 4.5 to 4.6 should not require much work.

A simplified class diagram of the project is presented in figure Figure 3.1. This diagram only details the most important features of each class, namely methods and properties, while keeping inheritance and dependency information intact, thus presenting the whole project in a meaningful yet compact form.

3.1.1 Applications and Gesture Recognisers

Qt's plug-in infrastructure (Nokia Corp., 2009) requires an interface to be declared in order to know how to load and link objects, and concrete plug-ins need to implement that interface.

IApplication is the interface to develop Doodle Applications. These applications are the building blocks of a tangible tabletop application based on Doodle. Examples of Doodle Applications include waveform generators, LFOs, band-pass filters, or anything else that needs to interact with other Doodle Applications in order to manipulate their behaviour and gather information from the environment, for example from gesture recognisers.

A Doodle Application may declare three lists.

Tangible objects: it is the list of the Tangible objects the application wants to manage and monitor. For example, the Tonalizer application declares the Tonalizer Tangible object.

Gesture Recognisers: it is the list of the gesture recognisers whose results the application is interested in. For example, the Sequencer application declares MusicTouch, MultiTap, and StraightStroke.

Widgets: it is the list of the Widgets the application is supposed to provide for visual feedback. For example, the Sequencer application declares KeyboardWidget.

These lists contain the explicit names of the objects. This way Doodle can automatically

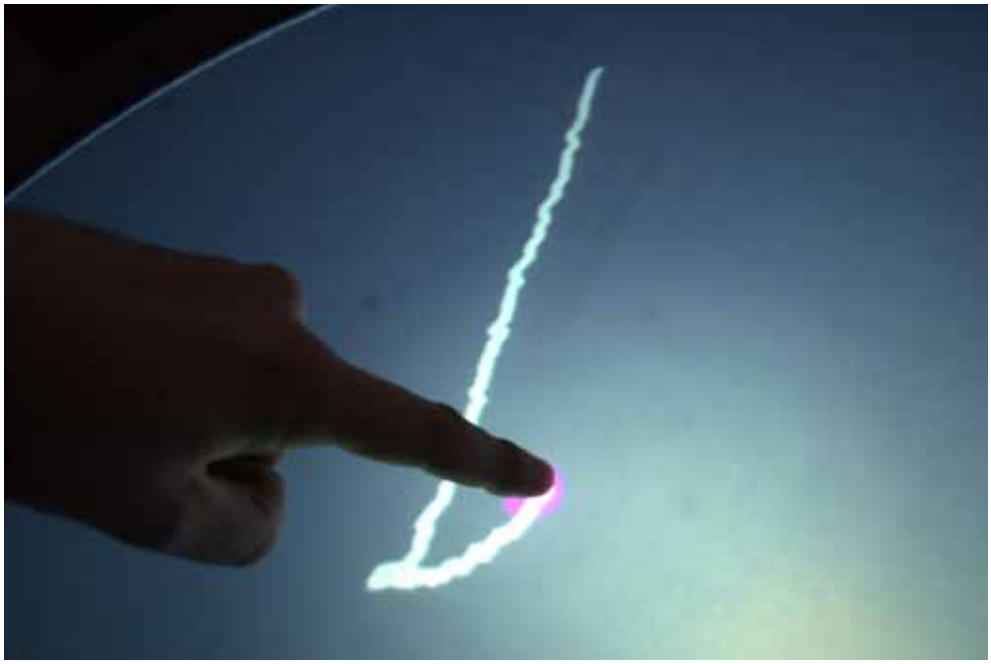


Figure 3.2. a single Trace made of a single Stroke

associate default signals and slots[1] every time a component enters the system. A Doodle Application features slots for processing results from gesture recognisers, messages from other applications and any event concerning Tangible objects, and it sends messages to other applications by emitting signals itself.

IGestureRecogniser is the interface to develop gesture recognisers. When a Group of Traces (section 3.1.2) is finalised, it is fed to all the gesture recognisers available for processing. When the processing phase ends each gesture recogniser may emit a signal with the result – or may even not, it depends on single cases, for example if the recognition phase fails. At this point, all the Doodle Applications that requested a particular gesture recogniser receive the emitted signal and start their processing phases. Gesture recognisers also have priorities that can be configured through the main XML file. It is not mandatory for applications to follow these priorities, though. In fact, single applications can even decide to completely ignore them and implement their own priority scheme. Last but not least, it is worth noting that a progressive gesture recogniser is continuously fed with Groups every time they are updated (that is every time one of their Cursors are updated).

1. For an overview on signals and slots in Qt see (Nokia Corp., 2009).

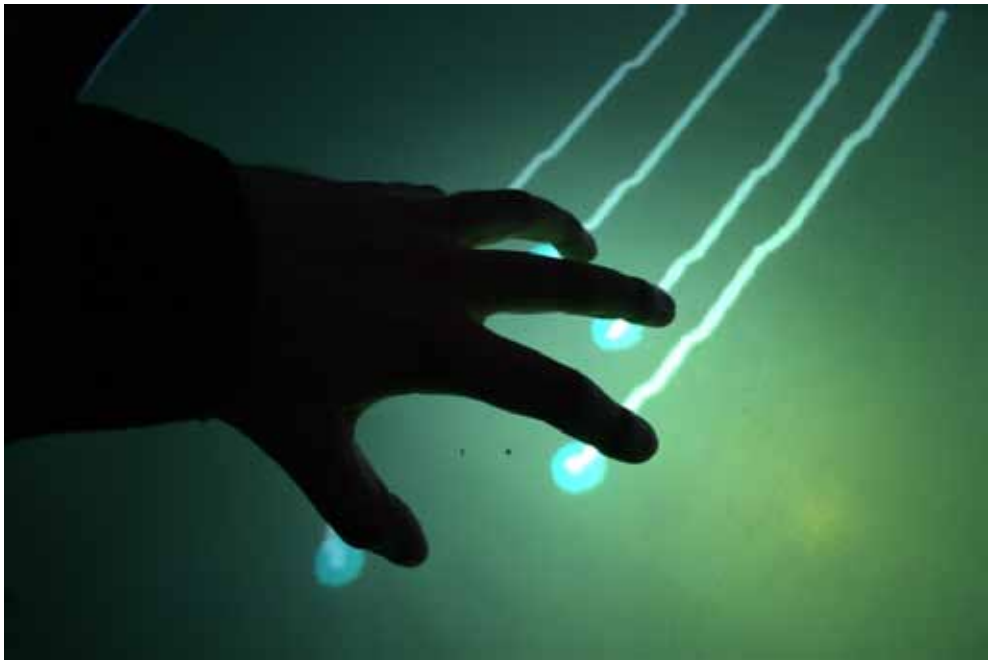


Figure 3.3. a Group of Traces

3.1.2 Tangible objects and finger tips

TUIO protocol describes tangible objects and cursors as targets. The TUIO C++ library provides an interface that receives such objects from an UDP stream and converts them to C++ objects. These objects are then passed to client applications as pointers. A major issue that emerged during development is that, at apparently random times, these pointers seemed to vanish, thus producing unwanted effects like dangling pointers and null references. This was reported to author Martin Kaltenbrunner, who carefully verified the sources and confirmed that some bugs were indeed present. Unfortunately, even after these bugs were corrected, the same misbehaviour occurred, even if at an extremely lower rate than before. For all practical purposes, it was decided that the best solution was to clone these objects as soon as they were available and make them locally available to Doodle until they were no longer necessary. This effectively solved the problem of dangling pointers.

Class Target is a prototype from which both classes Tangible and Cursor inherit basic properties, extending them where needed. For example, Tangible objects have a fiduciary ID attached to their bottom, while Cursors have a finger ID as specified by TUIO protocol. While motion of Tangible objects is of no particular interest in this case, and so its evolution can be tracked in real-time and then discarded, fingers are typically used to draw strokes that form complex gestures that need to be analysed both in real-time and at completion time. For this reason a set of classes has been developed.

Figure 3.4. two Traces each belonging to a different Group, as shown by the different colours of the Cursors.

Stroke: this class records a single Cursor motion, from the moment it appears to the moment it dies. Each point of the stroke is a TUIO Cursor, and so it has all the relevant properties like position, speed and acceleration.

Trace: at times, a finger can move so fast that reacTIVision loses track of it. A Trace is a sequence of Strokes used to close gaps between them. Two Strokes are considered to be related when the first point of the second Stroke is nearby the last point of the first Stroke. Also, these two points need to be close in time, otherwise the Trace is considered no longer active and cannot be resumed anymore.

Group: complex gestures are made of more than one related Traces, and a set of such Traces is a Group. Two or more Traces are said to be related when, for example, they appear nearby to each other or they're over the same visual target.

3.1.3 Visual feedback

Although Qt Framework 4.5 did support QWidgets to be painted into a dynamic texture, which is used in the Viewport of Doodle as we'll see in the section 3.2, it still didn't support multiple cursors and complex gestures, so direct multi-touch interaction with QWidgets was not possible. It was then decided to develop a set of Widgets with such characteristics. It shall be noted that these Widgets are QObjects, so they comply with Qt's guidelines, thus making it easier to port DoodleSDK to Qt 4.6. It's worth noting

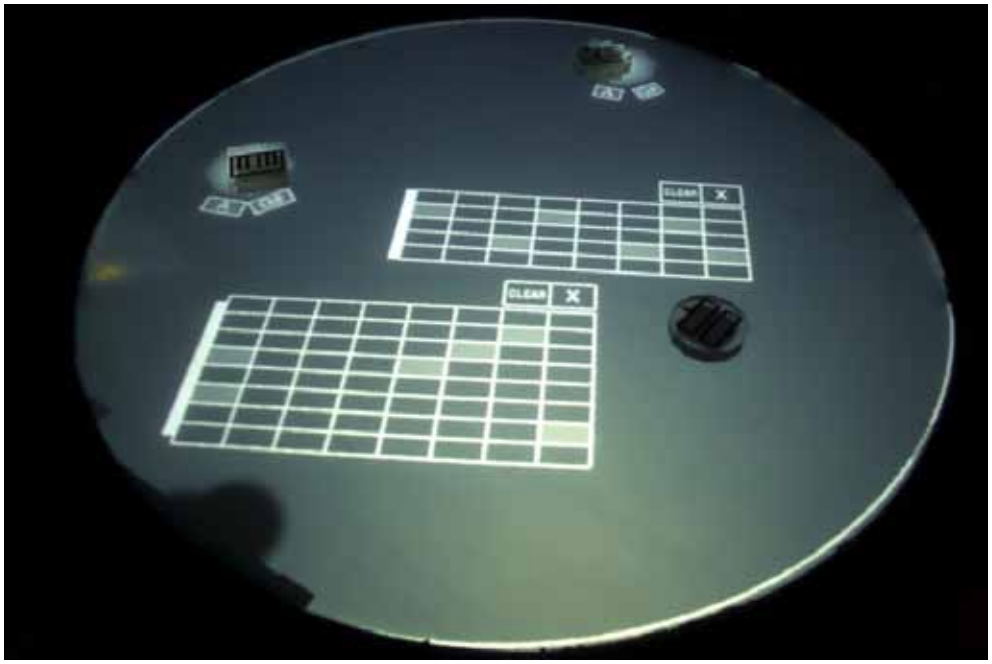


Figure 3.5. the Doodle demo application in action.

that QWidgets support custom painting and custom slots, so it is possible to create new widgets with custom appearance and behaviour simply by extending the QWidget class.

In addition to the Doodle::Widget base class, a small number of common Widgets were developed, namely LabelWidget, PushButtonWidget and CheckButtonWidget, whose properties and behaviour are documented inside the code.

Last but not least, DoodleSDK has two more utility classes.

OSCHelper: provides support for sending messages using the OpenSoundControl communication protocol. OSC messages are streamed over UDP using the oscpack C++ library. This is available to both Applications and Gesture Recognisers, though the communication mechanism based on QStringLists is preferred for internal communications, and OSCHelper should be used to communicate with applications that are external to Doodle. For example, the Sequencer application uses it to send MIDI messages to an expander developed in Pure Data.

Settings: provides support for storing configuration data read from an XML file. This is mainly used by the main Doodle Application described in the next section.

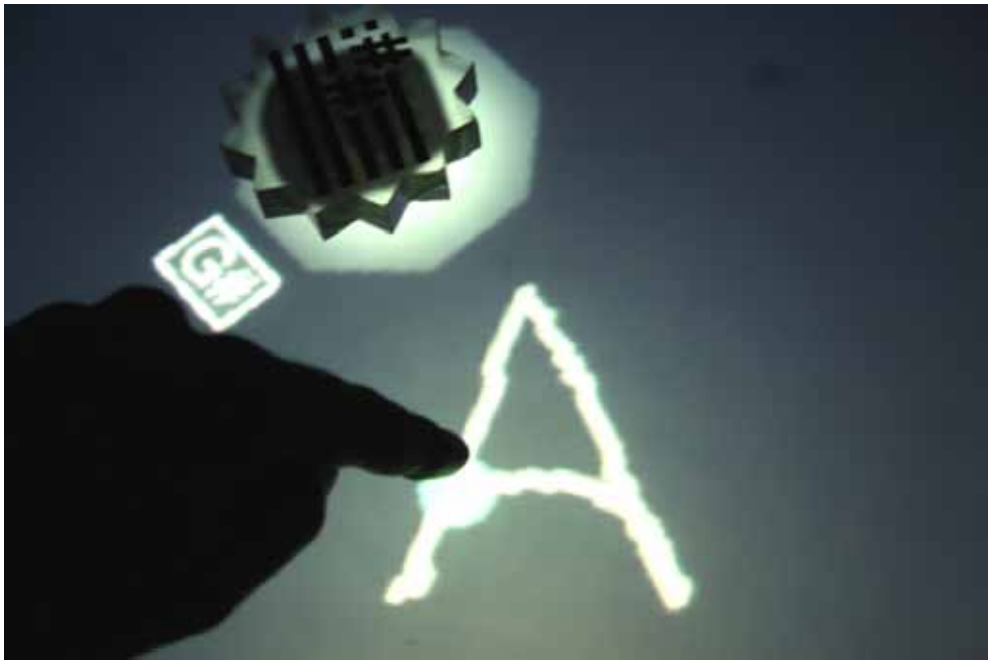


Figure 3.6. "A" glyph being drawn. As can be deduced from the source code of the Tonalizer Application, correct interpretation of this gesture will result in creation of a new slot for chord A major.

3.2 Doodle

Doodle is the actual core of the demo application. It is composed of an Application class, which is a special IApplication that acts as a supervisor and message router for all the other IApplications. This is the reason because the Application class in figure 3.1 both implements and depends on interface IApplication. This special Application is also a client for TuioListener, as indicated by the presence of the tuioCycle() method. This means that it directly receives TUIO messages, clones them (using the TuioProxy) and makes them available to all the other Applications and Gesture Recognisers.

Doodle also implements visual feedback. It may have been an interesting idea to develop a separate application that receives appropriate OSC messages from Doodle and renders the visual feedback, which is what happens in the original Reactable application, but for sake of simplicity it was chosen to make Doodle directly render the visuals.

For this reason the Viewport and Painter classes exist, although not detailed in figure 3.1. Painter class is a QPainter helper that's responsible for rendering all the visual objects – i.e. Widgets, Traces and other feedback – to a dynamic texture that is applied to a GL_QUAD eventually rendered inside the Viewport. The Viewport itself is a QGLWidget, which is a QWidget for displaying OpenGL content. It may seem

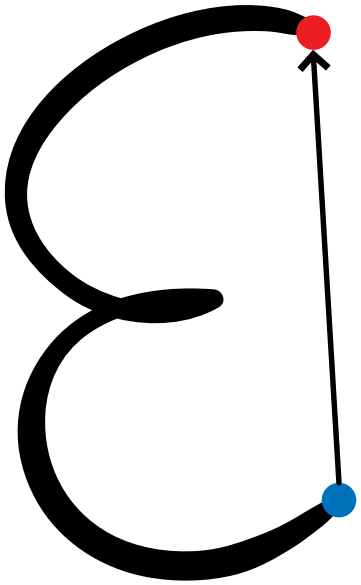


Figure 3.7. Bézier glyph for letter “E”. The red point is the first point, the blue point is the last point. The arrow represents the direction information associated with the glyph.

unreasonable to paint on a dynamic texture instead of directly in the Viewport unless noted that the beamer projecting the visual interface is not perpendicular nor perfectly aligned with the projection surface. This way it is possible to adjust the GL_QUAD to compensate for displacement and perspective distortion.

3.3 Glyph

Rather than implementing a particular recognition method, this gesture recogniser features a plug-in architecture so that many different handwriting recognition methods can be developed using only one gesture recogniser. Which method is to be used is set in the Doodle configuration file. This means that it's necessary to re-configure and restart the application in order to test another method, unless some way live re-configuration method is developed. Despite this apparent complexity, having a single broker that handles communication with Doodle dramatically reduces development errors. It also allows to develop recognition methods independently from Doodle, since the IGlyphEngine interface expects a QList of QLists of QPoints[1] instead of a Group.

1. It's a QList< QList< QPoint> > in C++ notation.

3.4 Glyph Recognition Engines

A number of techniques for handwriting recognition exist, from simple correlation of images to complex heuristics and handwriting movement analysis. Given the nature of user's input on the Reactable, given that interaction should be as easy as possible, and given that the set of symbols to be recognised could be large and dependent on different cultures, languages and even notation, it was almost natural to think of simple stylised symbols, like the glyphs that most PDAs and other similar devices are already implementing as their input system.

Considering that user's input is a set of points forming segments and curves, two different techniques were implemented.

3.4.1 Bézier engine

This is the first Glyph plug-in that was developed. It models user's input with a high degree Bézier polynomial and tries to match it with one of the models provided via a XML configuration produced using the Bézier Glyph building tool, which is documented at the end of this Chapter. A sketch of the matching algorithm is provided.

1. Take the first Glyph. Note: in current implementation, it actually takes just the first Trace of the Glyph. This is because the models to match against are made of a single Bézier curve. It should be trivial to extend this to examine all the Traces.
2. If necessary, decimate the number of points to less than 52.
3. Let n be the number of points - 1 after decimation. Use them to make Bézier curve of degree n.
4. Compute N constant time samples of user's Glyph.
5. Normalise the constant time samples inside the unitary square.
6. Compute the orientation of the Glyph. Note: orientation is defined as the vector starting at the first point and ending at the last point of the Glyph.
7. Load the models. For each model, rotate them according to input's orientation and compare one by one the points, one from the model and one from the Glyph to be recognised. Keep count of how many such pairs are within a given distance to each other. The model with the higher score is the best match.

Step 2 may sound strange, yet there is an explanation. The Bézier curve of degree n can be generalised as follows. Given the points $\mathbf{P}_0$, $\mathbf{P}_1$, … , $\mathbf{P}_n$, the curve is

$$\mathbf{B}(t) = \sum_{k=0}^{n} \binom{n}{k} (1-t)^{n-k} t^k \mathbf{P}_k$$

and the problem resides in the binomial coefficient. In fact $C(52,26)$ is the highest binomial coefficient that can be stored in 64 bit unsigned integers. For this reason it is necessary to work with curves of degree less than 52, unless bigger integers are available.

Reduction is done by subsampling the set of points at an appropriate ratio. More sophisticated techniques can be used, but this is safe and quick enough.

Step 4 also needs an explanation. The algorithm compares the first point from the model with the first point from the input, then the second from the model with the second from the input, and so on. Hence it is necessary to require models and input to have the same amount of points. Moreover, since a model's score is increased each time two such points are close enough, it is reasonable to make these points have the best chance to be close to each other. By sampling the equation at regular t intervals, the two corresponding samples are likely to be quite close to each other when the input and the model are similar enough. A constant cord sampling is also feasible but it is computationally more expensive and should have no particular advantages.

Step 6 is arguably the key point in making the approach even more natural, from user's perspective. In fact, each model has an associated direction information, which is given by the vector starting from the last point in the glyph and pointing to the first point in the glyph, as depicted in figure 3.7.

Provided users draw glyphs in the same way models are drawn, which is usually a requirement in similar systems like Palm's Grafiti or Xerox's Unistrokes, it is possible to align user's input with models, thus allowing users to draw glyphs in any direction.

3.4.2 Polygonal engine

As it is easily guessed, the Bézier technique is quite heavy, computationally speaking. In principle, this second technique is faster than the previous, but it eventually turned out to be less general and less precise in practice. A sketch of the algorithm is provided.

1. Take user's input, which is assumed to be drawn on a PadWidget.
2. For each model, try to match user's input with a model by subsequently rotating it by 90 degrees each step. Note: this way we make four comparisons using the same model because we don't know in which direction the user has drawn the glyph.
3. The model that contains the highest number of input's points is the best match.

Although it seems computationally easier, flaws in this technique are clear. First of all, it requires a widget to determine orientation of user's input. Second, models are similar to that in figure 3.8: in order to effectively match small variations of the same glyph, they need to be thick, and this way a lot of false positives are returned. Third, although not requiring users to draw glyphs in a specific way, there's no easy way to determine the orientation of user's input, so a reference is necessary – like a widget – and at least four comparisons – or even more – are needed. Increasing the number of repetitions is likely to bring computational cost near to that of Bézier technique with no guarantee of better matching performance.

Nonetheless this engine was kept to show that different engines can be developed.

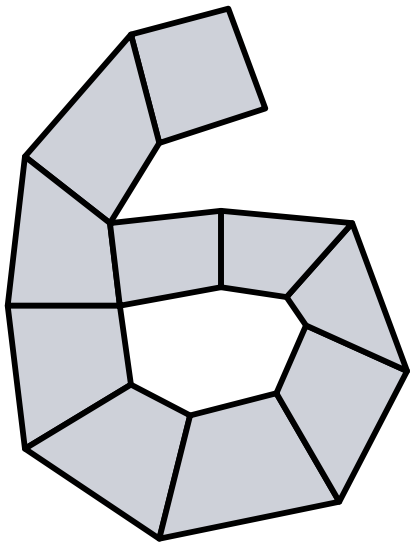


Figure 3.8. Polygonal model for number 6.

3.5 Application: Tonalizer

The Tonalizer application is the Doodle Application that implements ideas described in section 2.4.1. It's visually represented by a TonalizerWidget surrounded by ChordButtonWidgets, each of whose is used to represent a chord in the sequence of chords.

This application uses three gesture recognisers, namely Glyph, MultiTap, and StraightStroke. Glyph is used to create and edit chords in the sequence using handwriting recognition. The ChordWidget input method instead is only used to edit existing chords, and it's activated by dragging a chord slot away from the object, a gesture which is recognised by the StraightStroke gesture recogniser. Finally, MultiTap is used to interact with the ChordWidget. In fact this widget is a collection of buttons, both push buttons and check buttons, and can be activated, deactivated with one tap, and hidden with two taps. MultiTap is also used to activate and deactivate single chord slots, thus modifying the chord sequence.

This application also features an optional PadWidget which is used when a Glyph Recognition Engine needs a writing pad, like the Polygonal engine. While not being really useful on its own, this widget is still interesting because it's the only one that is actually pinchable, thus completely demonstrating the effects of the Pinch gesture recogniser which is otherwise only used to drag widgets around.

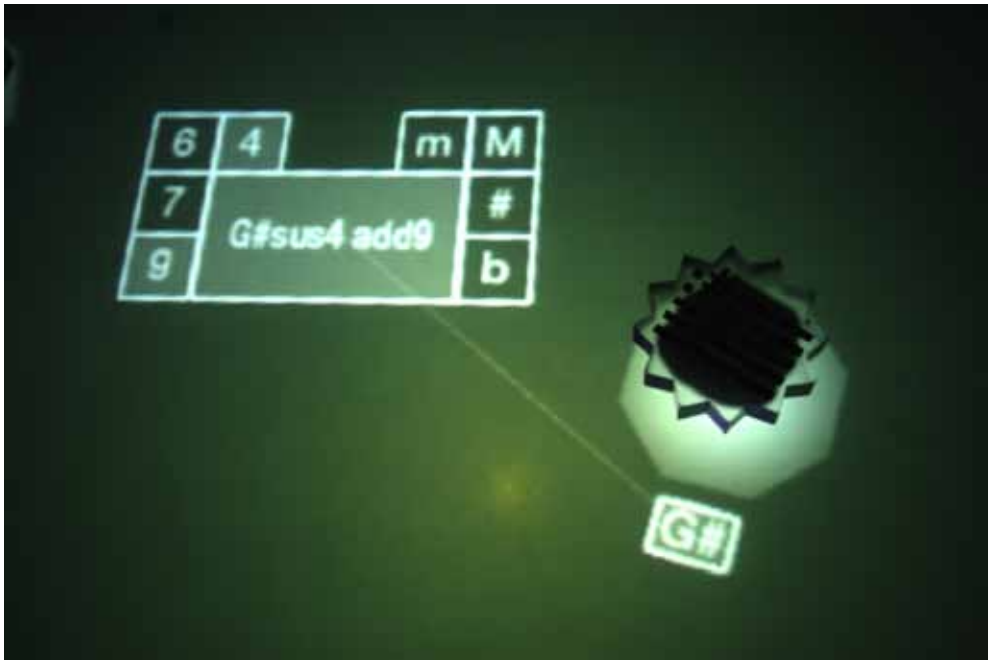


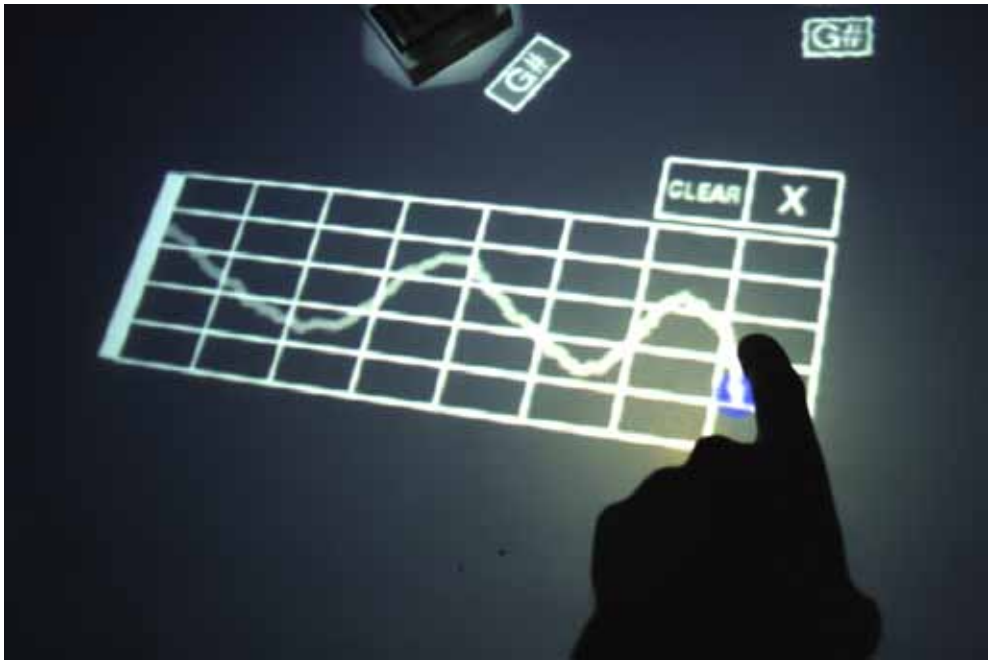
Figure 3.9. the Tonalizer with a chord slot and its ChordWidget.

3.6 Application: Sequencer

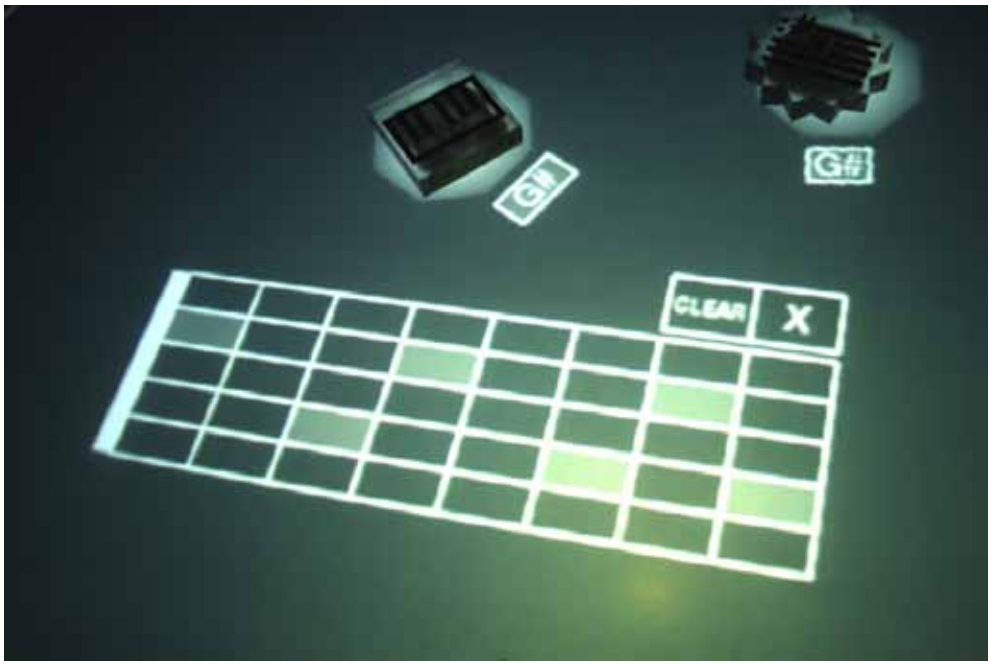
The Sequencer application is the Doodle Application that implements ideas described in section 2.4.2 by receiving information from the Tonalizer and creating appropriate sequence slots around the SequencerWidget.

This application uses two gesture recognisers, namely MusicTouch and StraightStroke. While StraightStroke is used to open KeyboardWidgets, in a way similar to that used by the Tonalizer to open ChordWidgets, MusicTouch is the gesture recogniser that turns strokes into notes on the KeyboardWidgets. These KeyboardWidgets are built according to the chord specified in the corresponding chord slot of the Tonalizer, and so they have a number of different scales that can be chosen, depending on those who fit the chord.

Which scales fit a chord is decided as follows. A large number of scales is set up when the application is first loaded. Note that these scales are lists of integers representing half-tone intervals from the root, hence for example a major pentatonic scale is [0, 2, 4, 7, 9] where 0 is the root, 2 is the major second, 4 is the major third, 7 is the perfect fifth and 9 is the major sixth. Each time a chord is created or modified, it is tested for inclusion in each scale. Those scales that entirely contain the chord are chosen. This is not a very sophisticated algorithm, yet it turned out to be quite satisfying when working with most scales, from the most common ones, like major and various minors, to less common ones like diminished or even whole tone and some other synthetic scales.



(a) MusicTouch sketch gesture



(a) MusicTouch result

Figure 3.10. MusicTouch gesture recogniser in action. In figure (a) a stroke is being drawn over a KeyboardWidget. In figure (b) we see the result of that gesture.

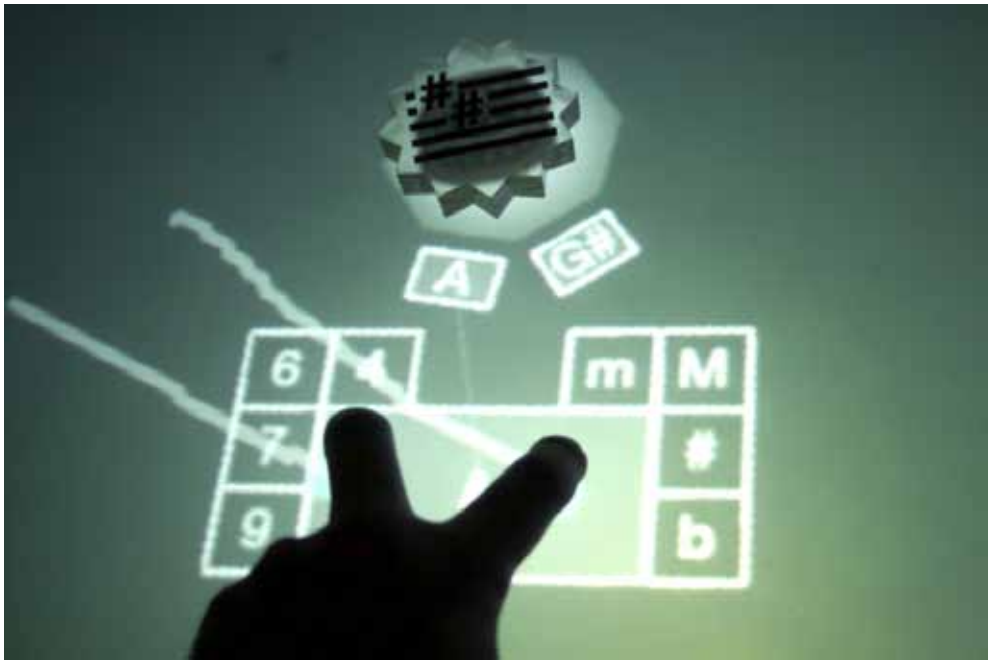


Figure 3.11. the Pinch gesture recognizer used to drag around a non-pinchable widget.

3.7 Application: Metronome

The Metronome application implements the object suggested in section 2.4.2. It is arguably the simplest application among those developed since its function is to broadcast a "beat" message at given time intervals.

The actual implementation is pretty straightforward: depending on the object's angle, a time interval is computed. This is fed to a QTimer object that calls the Metronome::beat() slot, which in turn emits a "beat" message. This message can be used by any other application interested in having a timing reference. For example, the Sequencer application uses the Metronome to regulate the advancement of the sequences and eventually play MIDI notes.

3.8 Other Gesture Recognisers

MultiTap: a tap gesture is like a mouse click. It is performed by briefly pressing a finger against the table's surface and lifting it shortly after that. This gesture recogniser detects multiple subsequent taps that are nearby in space in time. This is done by analysing all the Traces in a Group. If one of these is not recognised as a tap, then the Group is not recognised as a MultiTap gesture.

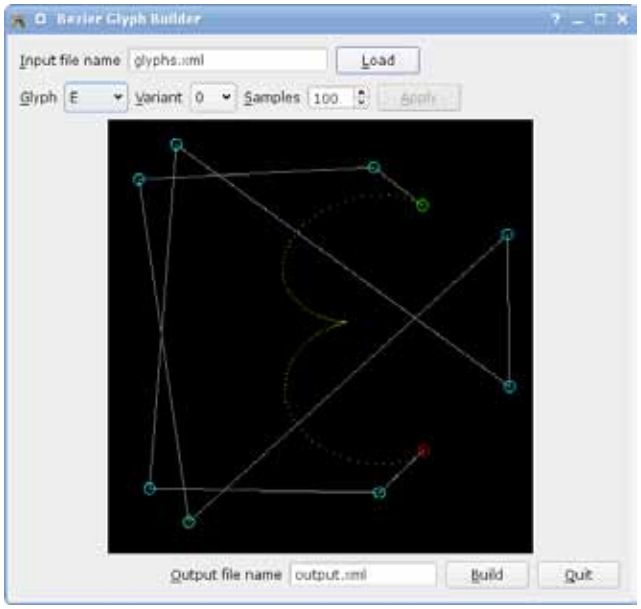


Figure 3.12. Bézier Glyph Builder.

Pinch: this is probably the most widely known multi-touch gesture. It is performed with two fingers that can approach or leave each other, thus performing a zoom action, or even be moved around together, thus performing a drag action, as in figure 3.11.

StraightStroke: as the name suggests, this detects whether a stroke is a linear segment or not. For a detailed description of the recognition method, see the class documentation.

3.9 Bézier Glyph building tool

The Bézier Glyph building tool shown in figure 3.12 is not strictly part of Doodle, while being an essential tool to construct the set of modules for the Bézier Glyph engine. It is an independent Qt application which allows to create and manipulate Bézier curves of arbitrary degree[1], save them to a source XML file and output the models – which are constant time samplings of the actual curves – to another XML file which will be used as a resource by the Bézier Glyph Engine.

The main interface is composed of two combo boxes for glyph/variant selection, and a drawing area in which control points of the Bézier curve being edited can be created, moved around, and deleted, and the resulting curve is previewed in real-time.

1. Actually, of degree up to 52, as noted in section 3.4.1.

The control points can be moved around by drag and drop, and they can even be moved around in groups. Multiple selections are performed by pressing the Control key and left-clicking on each control point. Creation of new control points is performed by selecting two or more existing control points and then pressing "A" on the keyboard. This will add a control point between each pair of selected points. Deletion is similarly performed by selecting the undesired points and then pressing "X" on the keyboard. The control points are rendered as cyan circles and the selected ones are white and thicker, but two special points exist, namely the first and the last. These are rendered respectively green and red, and they're used to determine the orientation of the glyph – which also means how the glyph is supposed to be drawn by the user, as explained in section 3.4.1.

When the glyph set is completed, it is possible to customise the number of constant time samples name of the output file, which is finally generated by clicking on the "Build" button in the lower right side of the window.

No similar tool was developed for creating a set of polygonal models. In fact, as the Bézier tool makes a lot of complicated work that is not feasible by hand, building a set of polygonal models is a lot simpler and can be done with very small effort by using any vector drawing software like Adobe Illustrator, Autodesk Autocad, free software like Inkscape and Blender, or even graph paper and pencil. In fact it's just a matter of drawing the polygonal mesh on a planar grid and writing each point's coordinate in an XML file which will be used as a resource by the Polygonal Glyph Engine.

4

Conclusions

4.1 Results

Due to lack of time and infrastructure availability, a thorough evaluation of the system has not been possible. However, each iteration presented in Chapter 2 was preceded by an informal assessment phase in which each draft has been submitted for evaluation to people familiar with the technology or at least familiar with the basic HCI concepts, as well as to random people whose knowledge level of these topics is nonexistent or at least not known. Comments and reactions of people involved in each assessment phase were noted and used in the next iteration.

The final iteration described in section 2.5 is the one that has been implemented in the final demo application that we described in Chapter 3. A sample of those who participated in previous assessment phases had access to an informal test session once the application was deemed sufficiently ready and stable. There had been no time to design in advance these test sessions in order to test specific features of the application by having users perform some precise goals and noting the results in a scientific way. Nonetheless all the sessions were aimed at evaluating how easy or difficult it was to execute some basic tasks, and whether users felt that a specific task was already easy enough to perform or presented any difficulties.

Although this final round of tests was not methodically carried on, it reported mixed yet interesting results.

- Most of the people who were already familiar with the Reactable, and so with the old Tonalizer, regarded the newly developed system as an interesting development, mostly because of the possibility to choose a subset of notes that ensures that no mistakes are made; however they also doubted that this whole renewed Tonalizer could really add some significant value to the Reactable as an instrument. On the other hand, some of those who weren't familiar enough with the Reactable didn't always get how this was an improvement at all, being just more fascinated with the original Reactable and its

sound exploration freedom.

- Regarding the overall simplicity of task performance, most of the people – both familiar and not – reported that some actions weren't that obvious to perform, for example the gesture that opens the chord widget or the piano-roll.
- They also reported that the reason because some slots around the tangibles were filled or empty was not really clear, although finding it reasonable when told. This suggests that more expressive visual feedback may be developed to improve feedback understanding.
- Almost everyone found that the piano-roll didn't always work as expected. This was absolutely expected due to the rough implementation of the algorithms.
- Nonetheless almost everybody found the handwriting idea pretty interesting in perspective, even funny, although it didn't always work as expected, but this is again due to rough implementation.

4.2 Future developments

The aforementioned observations suggest a number of improvements. Nonetheless there are other ideas, even not strictly related with this thesis, that may be worth exploring.

First of all, as noted in subsection 2.4.2, there is a problem with how to manage the choice of a different scale while a melody is already placed on the affected piano roll. There we discussed two possible solutions, although neither of these is currently implemented. Those two solutions shall be implemented and assessed in order to find out which one better fulfils the principle of least astonishment.

Speaking of multiuser, multi-touch and tangible interfaces, the range of possibilities they give about sound and music control is quite wide, yet they lack some important gestures like tapping or pushing (Bosi, 2009). It may be interesting to bring more different gestures to the Reactable, for example tapping and pushing, or even aerial gestures, like those controlling theremins, thus augmenting freedom of expression.

When accounting for multiple users, even if the system can be simultaneously used by a number of different people, it may be interesting to make it interact with other similar devices, or even traditional musical instruments. This would allow the creation of some sort of "virtual improvisation space" in which any kind of sophisticated technology can be implemented, from telling other performers about what's going on with other devices or instruments, to some kind of automatic suggestion for moving on the performance based on past and present information.

In Chapter 1 we discussed music education, but this is obviously not the entire story. During the development of this thesis there had been contacts with people involved in education and assistance to people with disabilities such as physical handicaps or even

autism and learning disorders. Most of them were extremely fascinated by the Reactable as a tool to make these people approach music and possibly help them express themselves. Also there have been comments about how such a colourful musical instrument could be approached by synæsthetes, and even if such an instrument could help in diagnosis of these conditions.

4.2.1 A final note about music theory

Despite this work's title, the proposed objects don't seem to bring much "music theory" into the Reactable, nor it's clear how they should represent a "practical approach" to it. The work developed for this thesis was heavily time-constrained, and as such it couldn't cover the entire spectrum of possibilities that a topic of such magnitude should require.

In section 1.5 we discussed implicit learning and music education, also hinting how the system described in this thesis should be developed in order to effectively claim to be an approach to music theory. The demo application described in Chapter 3 doesn't implement those concepts yet, but this thesis should be considered as a first important milestone, and such concepts are going to be considered and thoroughly developed in the future.

Bibliography

Emmanuel Bigand, Philippe Lalitte, and Barbara Tillmann. *Sound to Sense, Sense to Sound. A State of the Art in Sound and Music Computing*, chapter 2. Logos, 2008.

Tina Blaine and Tim Perkis. The Jam-O-Drum interactive music system: A study in interaction design. In *Proceedings of the ACM DIS 2000 Conference*. ACM Press, August 2000.

Mathieu Bosi. Extending physical computing on the Reactable. Master's thesis, Universitat Pompeu Fabra, 2009.

Noam Chomsky. *Aspects of the Theory of Syntax*. The MIT Press, 1965.

Sergi Jordà, Gunter Geiger, Marcos Alonso, and Martin Kaltenbrunner. The reacTable: Exploring the synergy between live music performance and tabletop tangible interfaces. In *Proceedings of the first international conference on "Tangible and Embedded Interaction" (TEI07)*. Baton Rouge, Louisiana, 2007.

Golan Levin. Painterly interfaces for audiovisual performance. Master's thesis, Massachussets Institute of Technology, 2000.

Jean Molino. Toward an evolutionary theory of music and language. In Nils Lennart Wallin, Bjorn Merker, and Steven Brown, editors, *The Origins of Music*, Bradford books, pages 165–176. The MIT Press, 2000.

Nokia Corp. Qt 4.6 whitepaper. `http://qt.nokia.com/files/pdf/qt-4.4-whitepaper`, December 2009.

James Patten, Ben Recht, and Hiroshi Ishii. Audiopad: a tag-based interface for musical performance. In *Proceedings of the 2002 conference on New Interfaces for Musical Expression*, May 2002.

Nicolas Ruwert and Mark Everist. Methods of analysis in musicology. In *Music Analysis*, volume 6, pages 3–36. Blackwell Publishing, 1987.

Carol A. Seger. Implicit learning. In *Psychological Bullettin*, pages 115:163–169. 1994.