Max Huttunen

Tahti: a groovebox for the web browser

Master's Thesis

https://thesis.tahti.studio

Thesis Advisor
Esa Onttonen

Thesis Supervisor
Antti Ikonen

Aalto University
School of Arts, Design and Architecture
Media Lab Helsinki
Master's Degree Programme in Sound in New Media
March 2021

Aalto University
School of Arts, Design
and Architecture

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi

Master of Arts thesis abstract

Author Max Huttunen

Title of thesis Tahti: a groovebox for the web browser

Department Department of Media

Degree programme Sound in New Media

Year 2021 **Number of pages** 50 **Language** English

Abstract

This thesis describes the background, design decisions, and implementation details of a novel application for creating music in a web browser. The groovebox format is selected to limit the thesis's scope and because its constraints are believed to lead users to more interesting creative results compared to applications that are inflated with features. By examining existing browser-based applications resembling grooveboxes, it becomes evident that they are far behind their hardware counterparts in terms of features, usability and creative possibilities. The design presented in the thesis is formed by combining concepts from hardware grooveboxes with original ideas.

The implemented application uses audio samples as a starting point for generating sound. Each of the eight separate tracks features various sound processors for shaping the track's sound. All of the tracks also have a dedicated step-sequencer with advanced sequencing features to build intricate and evolving musical patterns. One of the key features is step modulation, allowing the modulation of almost any of the sound-shaping parameters on a per-step basis. The user interface is designed to be easily usable for knowledgeable first-time users while offering many advanced features to keep long-time users interested as well. Another design principle is to encourage serendipity, or the accidental discovery of pleasantly surprising results, which is achieved by providing randomisation functions and macro controls.

Although the produced artefact is already entirely usable, the application's development is still in its early stages. Essential tasks considering post-thesis work are improving the audio processing algorithms, building features for more efficient sharing of patterns, and improving the overall quality by finding and fixing bugs. An exciting possibility would be to bring the application to other platforms, like mobile phones, for even more accessible creation of musical ideas or to digital audio workstations for finishing compositions more practically. An important feature available on many hardware grooveboxes, but still missing here, is the ability to chain patterns to create longer, more evolving compositions. Nevertheless, despite some of its shortcomings, the application is already considerably more advanced than the selection of existing browser-based groovebox-like tools presented in the text.

Keywords groovebox, software design, sequencer, web browser, drum machine

Aalto-yliopisto
Taiteiden ja suunnittelun
korkeakoulu

Aalto-yliopisto, PL 11000, 00076 AALTO
www.aalto.fi

Taiteen maisterin opinnäytteen tiivistelmä

Tekijä Max Huttunen

Työn nimi Tahti: a groovebox for the web browser

Laitos Median laitos

Koulutusohjelma Sound in New Media

Vuosi 2021 **Sivumäärä** 50 **Kieli** Englanti

Tiivistelmä

Tämä opinnäytetyö käsittelee uudenlaisen verkkoselaimessa toimivan musiikintekosovelluksen taustaa, suunnitteluprosessia ja toteutusta. Sovelluksen tyypiksi on valittu groovebox työn laajuuden rajoittamiseksi. Groovebox-tyyppisten sovellusten perustellaan myös johtavan käyttäjiä kiinnostavampien lopputulosten äärelle verrattuna sovelluksiin, joissa on enemmän toiminnallisuuksia. Olemassa olevien selainpohjaisten groovebox-sovellusten tarkastelun myötä on kuitenkin selvää, että ne ovat ominaisuuksiltaan, käytettävyydeltään sekä luovalta potentiaaliltaan huomattavasti fyysisiä vastineitaan jäljessä. Opinnäytetyön esittelemä design on muodostettu yhdistelemällä fyysisten groovebox-laitteiden konsepteja tekijän omiin ideoihin.

Toteutetussa sovelluksessa on kahdeksan raitaa, joilla lähtökohtana äänen tuottamiselle toimivat valmiit ääninäytteet. Jokaisella raidalla on useita ääniprosessoreita raidan ääninäytteen reaaliaikaista muokkausta varten. Lisäksi jokaiselta raidalta löytyy askelsekvensseri, jonka monipuoliset ominaisuudet mahdollistavat hienovaraisten ja kehittyvien musiikillisten tuotosten luomisen. Eräs sekvensserin keskeisimpiä ominaisuuksia on parametrimodulaatiotoiminto, jonka avulla lähes jokaisen ääntä muokkaavan parametrin arvo voi vaihdella askelkohtaisesti. Sovelluksen käyttöliittymä on suunniteltu helposti lähestyttäväksi asiantuntevalle ensikäyttäjälle, mutta se tarjoaa kuitenkin myös useita yksityiskohtaisia toimintoja edistyneempienkin käyttäjien mielenkiinnon ylläpitämiseksi. Toinen olennainen suunnitteluperiaate on onnekkaiden sattumien edesauttaminen, mikä on toteutettu tarjoamalla satunnaistamistoimintoja sekä makro-parametreja.

Vaikka sovellus on jo täysin käyttökelpoinen, on sen kehitystyö vielä alkuvaiheessa. Sovelluksen jatkokehityksen kannalta olennaisinta on parantaa äänenkäsittelyalgoritmeja, helpottaa tuotosten jakamista käyttäjien kesken sekä kohentaa sovelluksen yleistä laatua etsimällä ohjelmointivirheitä ja korjaamalla niitä. Fyysisiin groovebox-laitteisiin verrattuna tärkein puuttuva ominaisuus on pidempien musiikillisten kokonaisuuksien luominen. Sovelluksen tuominen muille alustoille, kuten älypuhelimille tai digitaalisille äänityöasemille, on sen sijaan yksi mielenkiintoisimmista pitkän tähtäimen jatkokehitysmahdollisuuksista. Puutteistaan huolimatta toteutettu sovellus on kuitenkin jo huomattavasti kehittyneempi kuin työssä esitellyt olemassa olevat selainpohjaiset groovebox-sovellukset.

Avainsanat groovebox, ohjelmistosuunnittelu, sekvensseri, verkkoselain, rumpukone

Acknowledgements

First, I would like to thank Esa Onttonen for his invaluable help with the text and with bringing the thesis to a conclusion. I would also like to thank Tommi Gröhn for guiding me in the right direction in regards to some of the audio algorithms, most notably the reverb. Finally, I would like to express my gratitude to SOUL's development team, namely Cesare Ferrari and Julian Storer, for being incredibly responsive to my bug reports and always fixing them promptly.

Table of contents

1. Introduction

1.1. Background and motivation

The term ‘groovebox’ is most often used to refer to multi-track hardware instruments that incorporate sound-producing mechanisms with step-sequencers to produce entire musical compositions. The term was first used by Roland to describe its MC-303 instrument which was released in 1996 (Future Music, 1996, p. 24), and has since been in frequent use by various manufacturers with new hardware grooveboxes still emerging regularly. There is no strict definition for what constitutes a groovebox, and the term is often used interchangeably with ‘drum machine’. However, in this thesis, the term ‘groovebox’ is used throughout to avoid the connotation that such a device would only be useful for the creation of drum sounds.

In the software world, products describing themselves as grooveboxes are almost always found in mobile applications, but hardly ever on desktop systems. Instead, digital audio workstations (DAWs) are the de facto way of creating music on desktop computers in tandem with plug-in software. In such setups, the DAW is in charge of the sequencing while the plug-ins take care of producing and altering sounds; it is a modular environment with very few limitations on what can be achieved. In contrast, grooveboxes are generally much more limited by providing only a fixed set of sound producing and processing options, and more primitive sequencing options. However, it can be argued that precisely these limitations are what create the appeal for the groovebox format: limitations spark creativeness as the user cannot get lost in endless possibilities and is forced to exert extra creativeness to make use of what is available. This notion is reflected by Magnusson (2010), who discusses the problems that infinite scopes of environments have on creativity, and Boden (1990, p. 95), who argues that constraints are what empower thinking. Building a groovebox for the browser, in particular, makes sense since applications running in the browser need to be self-sufficient: interfacing with other applications even on the same machine is arduous. In addition, like smartphones and hardware devices, browsers do not have evolved software ecosystems or enough performance for hosting serious competitors to native DAW software.

The advent of modern web technologies has enabled creating audio applications that run entirely inside the browser. As the web browser is the most widely used software environment in history, modern web technologies provide the opportunity for musical interfaces to be used by entirely new audiences (Roberts, Wakefield and Wright, 2013).

However, most browser-based music creation tools that resemble grooveboxes and not aiming to be fully-fledged DAWs are still very crude, and too limited to be seriously considered for music creation purposes. They do not come close to even the simplest of hardware grooveboxes. The reason for the primitiveness of existing browser-based tools should not anymore be caused by limitations in the technology, but perhaps building more sophisticated applications has so far been considered pointless.

There should not exist a reason why an advanced groovebox built for use in a browser could not be exciting and enjoyable to use. I turn to my grooveboxes whenever wanting to focus on creating new ideas without unnecessary distractions brought on by a DAW. However, these hardware machines are bulky and inconvenient to detach from their setup, essentially only getting used in one physical spot. With a browser-based groovebox, musical ideas could theoretically be started on any computer by simply visiting a link, even without having to explicitly download any software. This prospect becomes even more attractive when taking browsers on mobile devices into consideration. A browser-based groovebox could empower completely new user-bases to benefit from the central features and creative paradigms of a groovebox without having to invest in actual hardware. In addition, a groovebox on the web would inherently encourage collaboration and social interactivity. This is not to say that such functionality would emerge automatically, but in a web context, it would certainly be more straightforward and natural to implement than in the context of a native application, for example.

1.2. Objectives

The objective of the thesis is to design and implement a software application for innovative music creation that runs entirely in a modern web browser. The aim is to design an application that is comparable to hardware grooveboxes in terms of its features and creative capabilities, which should be achieved by taking inspiration from existing grooveboxes. In other words, the application should at least have multiple parallel tracks, enough audio processors for interesting sound sculpting possibilities, a variety of modulation options, and a user-interface that facilitates a pleasant user experience. The produced artefact should be easy for beginners to use while still having the necessary depth and complexity for keeping extended usage intriguing. Due to the state of current technologies, further elaborated in chapter 4.4, only a desktop interface will be developed, simultaneously limiting the scope of the thesis. Developing both mobile and desktop interfaces would be a tremendous undertaking as touch interfaces require completely different user-interface paradigms than desktop interfaces. Should the developed artefact not already be a usable application, it should at least act as a solid foundation for future post-thesis work.

1.3. Methodology

No strict development or design method or framework was followed due to the uncertain nature of the project. At the start, there was no way to be sure whether browsers would have the required performance to simultaneously run several tracks with dedicated processing. Therefore, the application had to be developed in small increments to figure out if performance would become a bottleneck and what kind of features would be feasible. First, just one track was implemented, after which the number of tracks was slowly increased while also implementing more per-track processors. The selection of included per-track processors had to be carefully balanced to keep track count at the aimed number of eight. For example, an earlier version with four tracks featured a per-track delay effect, but this design had to be discarded because it was simply too processor-intensive when the track count was increased. For conceptual design decisions, inspiration was drawn from the hardware grooveboxes presented in chapter 2.2., but also novel ideas that have arisen from personal experience were leveraged. In contrast, existing browser-based tools presented in chapter 2.1. proved to be less useful for fueling new ideas due to their elementary features.

An early 2015 Macbook Pro with a 2.9 GHz processor was used as the baseline for testing performance because it was the least powerful machine I had available. However, it is still a high-end computer, and I wanted the application to be usable on more basic devices as well. Consequently, to simulate the resources of a less-powerful computer, any features that resulted in using more than half of the processor's resources with the Chrome browser were deemed too heavy. While certainly not a scientific method, this measurement seemed suitable considering the scope of the thesis. Other browsers, including Firefox, were also briefly tested to ensure that general functionality worked, but their performance was not benchmarked as closely.

1.4. Structure of the thesis

The main content of the thesis is divided into three major chapters. In chapter 2, the features and capabilities of most existing browser-based music creation tools incorporating a step-sequencer are listed and examined. In the same chapter, a few hardware grooveboxes are inspected, which act as the primary source of inspiration for the features implemented to improve on existing browser-based designs. Chapter 3 begins by discussing some of the fundamental design principles, followed by detailed descriptions of the design and implementation of different parts of the application. Finally, in chapter 4, future improvements are discussed, followed by evaluation and concluding remarks in chapter 5.

1.5. Terminology

Groovebox

A groovebox is a self-contained electronic instrument mostly found in hardware-form that can be used to create loop-based electronic music using sampling, synthesis or both. Grooveboxes usually have multiple tracks and integrated step-sequencers, sometimes even making possible the creation of full songs. The definition of the term ‘groovebox’ is inexact, and it is often used interchangeably with the term ‘drum machine’.

Step-sequencer

A sequencer represents timed musical events, allowing them to be edited and played back at a certain speed, or tempo. Step-sequencers allow setting these musical events on a fixed number of steps, which are evenly placed in a sequence. The most common amount of steps is 16 (and multiples of it) since it perfectly represents 1 bar of 16th notes in a 4/4 signature.

Track

In the context of sequencers, tracks describe different overlapping lanes of musical events. When a sequencer is running, events in all of the tracks are superimposed. Different sounds are usually arranged to be played on different tracks. For example, a snare and hi-hat could have their triggers arranged on separate tracks.

Voice

A voice encapsulates the set of audio generators and processors used to produce a sound in an audio application. In grooveboxes, each track is usually associated with a voice, although there are cases where multiple voices are shared across tracks to allow polyphonic playback. In a groovebox, a voice could contain a sample player, filter, and amplification envelope, for example.

Parameter

In this text, numerical values representing some sonic properties are referred to as parameters. Parameters have a value and a range that must constrain the value at all times. For example, the pitch of a played tone could be controlled by a parameter with a range from -24 to +24 semitones. As the parameter is the abstract concept behind a sonic property, the control is the actual physical or virtual user-interface element that allows a user to change the parameter's value. Controls are usually in the form of knobs, toggles and sliders.

Modulation

Modulation is used to describe changing parameter values over time. Modulation can be in the form of an envelope, a low-frequency oscillator (LFO), or programmed data on a timeline, to name just a few approaches. Step modulation refers to a paradigm where modulation of specific parameters can be applied separately to each step on a step-sequencer.

2. Existing work

In the first part of this chapter, the text examines existing browser-based applications resembling grooveboxes and compares the tools in terms of their features and creative possibilities. Although a slew of browser-based DAWs also exists, these are not explored as the focus in this thesis is to develop an instrument more resembling a groovebox. After that, a selection of hardware grooveboxes is analysed similarly. Finally, the state of existing tools is analysed based on the findings.

2.1. Browser-based tools

The selection of browser-based applications presented in this chapter represents the state of existing web applications related to grooveboxes in their general mechanics. The following is not a complete list of every groovebox-esque tool found online, but it is a healthy representation of the overall character of available applications; the presented applications cover most of the features found in browser-based grooveboxes in general.

2.1.1. Roland 808303.studio

808303.studio is a rudimentary emulation of the classic Roland 808 drum machine and the Roland 303 bassline synthesiser. The drum machine's sequencer consists of 16 steps per sound with control over the tempo and shuffle. Individual sounds cannot be edited, and no options for creating modulation or variations are present. The original hardware drum machine was not too complicated either, but it still featured a few controls per sound and programmable accents in the sequencer. Since Roland and the Design Museum played a part in the creation of the Roland808303.studio (*Roland 808303.studio*, 2020), the purpose of the work appears to be more of an homage to the original machines as opposed to a useful and functional tool to create music in the browser.

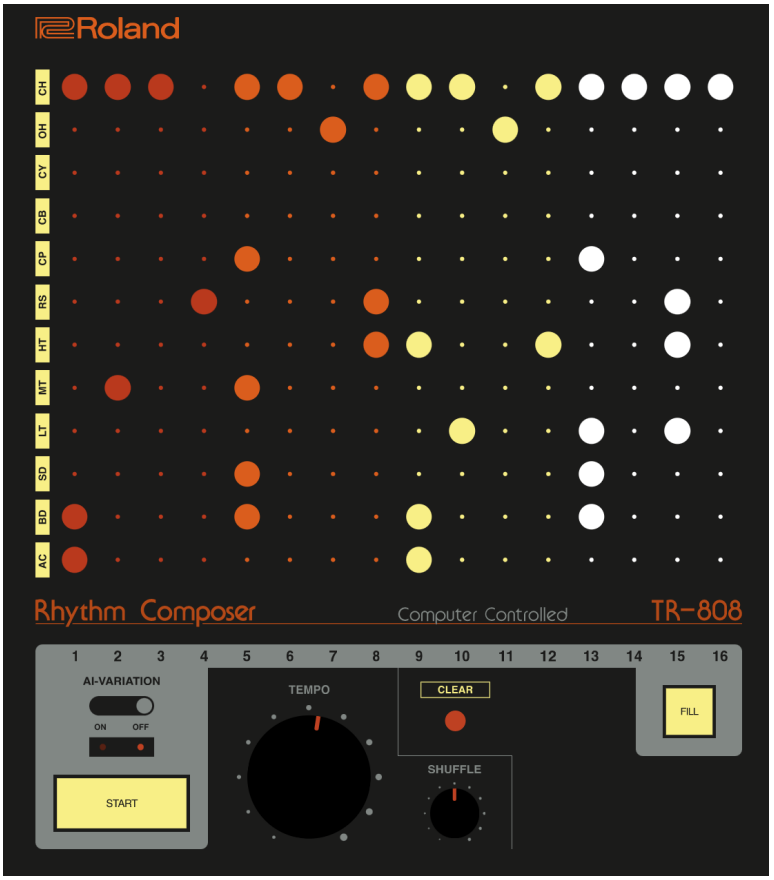

Figure 2.1. Screenshot of Roland 808303.studio

2.1.2. Drumbit

Feature-wise, drumbit is much like the 808303.studio, but Drumbit offers slightly more functionality. The application is a sample-based drum machine with a 16-step sequencer and eight tracks. The used samples are only changeable by switching to a different predefined kit which will change the sounds of all eight tracks at once. However, a premium version is available for purchase with support for user samples (*drumbit - online drum machine*, 2017). The volume and pitch for each sample are adjustable statically, and globally adjustable parameters include the tempo, amount of swing, master volume and the used effect. Available effects include a reverb with preset configurations and a configurable low-pass filter, high-pass filter, and compressor. Although a bit more flexible than Roland's 808303.studio, Drumbit also lacks ways to create more intricate patterns or to edit the sounds even statically. The user interface is also quite bare; it does not encourage exploration, and interactions feel awkward at worst.

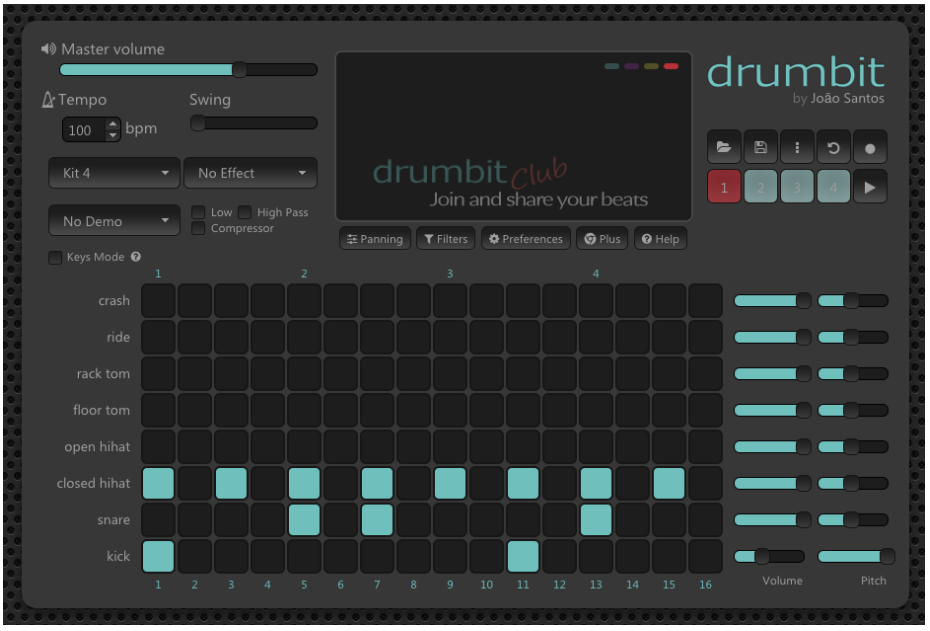

Figure 2.2. Screenshot of Drumbit

2.1.3. WebAudio Drum Machine

The WebAudio Drum Machine features very similar functionality as Drumbit, but it is unique in providing entry-level step-based variation. The application is part of a set of demos showcasing the possibilities of modern web audio APIs and it features six tracks on a 16-step sequencer with a dozen predefined drum kits. The controllable parameters include each track's levels and the global tempo and swing amounts. Many different processors, including reverbs and filters, can be selected as a master effect. Unlike many other web drum machines, the WebAudio Drum Machine can play samples at two different levels on a per-step basis. Such functionality can be viewed as very rudimentary step-based modulation of the volume parameter, which already allows the user to program considerably more nuanced rhythms than other applications. (Rogers and Moore, 2012)

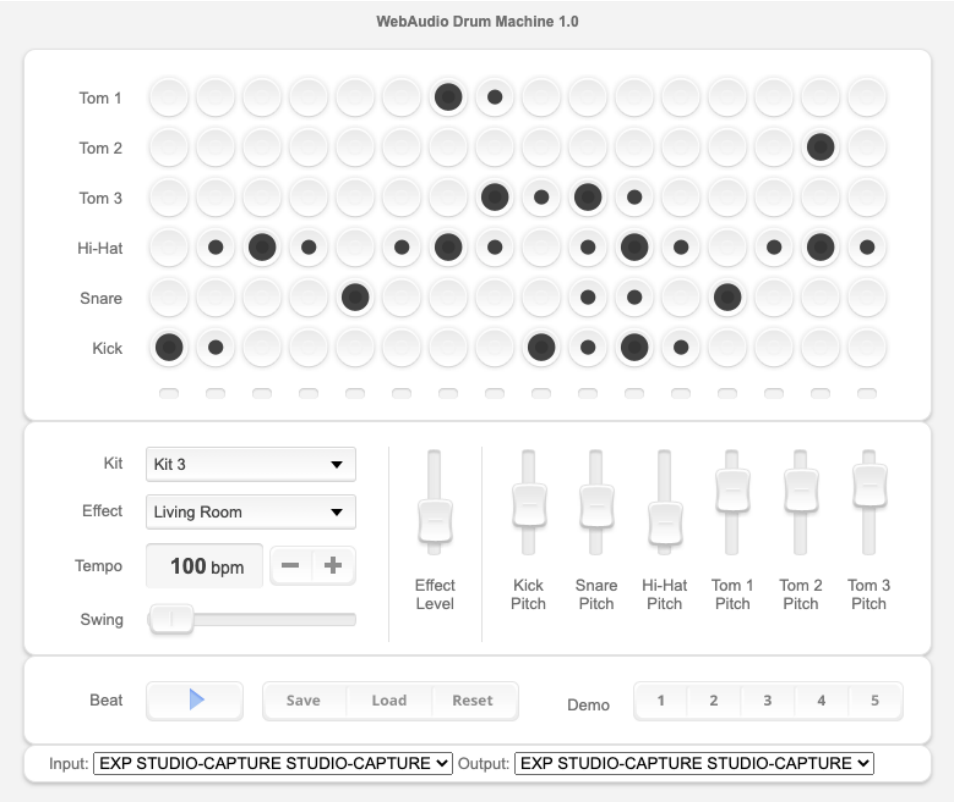

Figure 2.3. Screenshot of WebAudio Drum Machine

2.1.4. Splice Beat Maker

The Splice beat-maker is rare because it provides a specific service instead of being an entry-level experiment or proof-of-concept application. Splice is a subscription-based online service providing members access to an extensive library of royalty-free audio samples. At the time of writing, the sample library consists of over 2 million samples (*Splice sample search*, 2021), and members of the service also gain access to the online Beat Maker, an 8-32 step-sequencer with 16 separate tracks. Any sample from the Splice library can be used on each track in the Splice Beat Maker, which, due to the immense amount of samples, is useful for experimenting with how different sounds fit together before downloading them. Evaluating how samples sound together is Beat Maker's purpose, and it does not go further from being a simple prototyping tool: it offers no control over individual track parameters, not even the relative levels can be adjusted. However, the user-interface feels quite polished, and the tool fulfils its minimal purpose effectively. (*Splice Beat Maker*, no date)

Figure 2.4. Screenshot of Splice Beat Maker

2.1.5. Genius Home Studio

Genius Home Studio is arguably one of the most ambitious web-based sequencer tools. It is a sample-based beat-maker allowing users to create up to 32-bar compositions with 16 steps per bar (*Genius Home Studio*, 2020). No limit on the number of tracks is mentioned, but the tool becomes unresponsive after adding dozens of tracks. A sample from a set of default samples can be selected for each track, and sounds can also be uploaded from the user's computer. The only track-specific parameter that can be adjusted is volume, which can only be nudged by half-decibel increments via the interface. Master controls only include the tempo and position of the playhead. The user interface shows 16 steps at once, and getting an overview of more extended patterns is difficult since there is no vertical zoom. As with the previously examined tools, Genius Home Studio offers no modulation options or other features to fine-tune patterns on a per-step basis. Despite its seemingly ambitious purpose of facilitating the creation of full-length songs, the tool seems to be mostly a marketing stunt to promote HP laptops and the Genius website (*Announcing 'Genius Home Studio' Music Creation & Collaboration Program Powered By HP*, 2020).

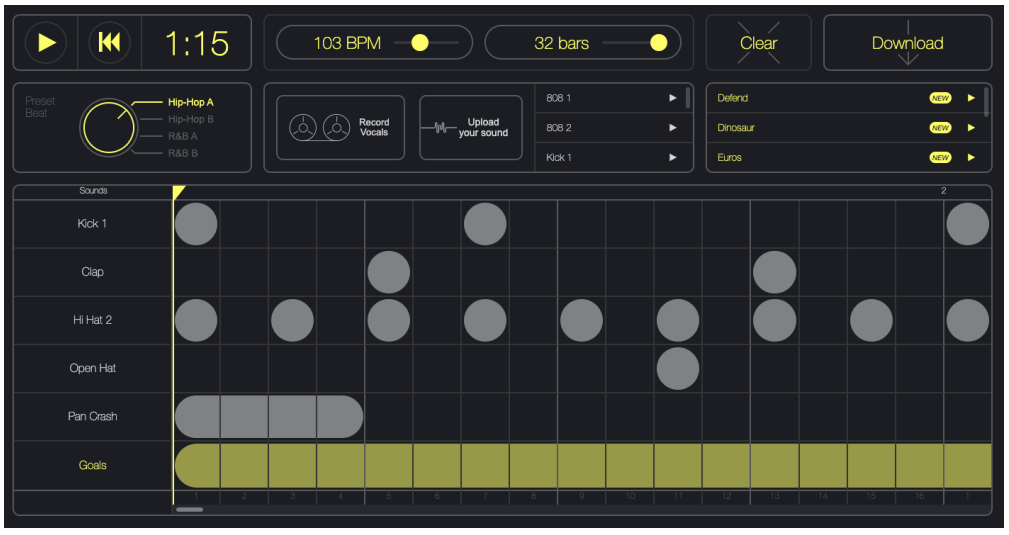

Figure 2.5. Screenshot of Genius Home Studio

2.2. Hardware grooveboxes

As grooveboxes have mainly existed in hardware form, the variety of different products is also vast. A pedantic look into all grooveboxes on the market would be an arduous undertaking, which is why only a few devices are explored in this chapter. The presented grooveboxes are selected based on my familiarity with them, and they are considered an appropriate sampling considering the thesis's scope. In the end, hardware grooveboxes often have a lot of overlapping functionality. For inspiring future work beyond the thesis, however, it might be useful to expand the exploration into other hardware grooveboxes as well.

2.2.1. Elektron grooveboxes

Elektron is a Swedish manufacturer of electronic instruments specialising in grooveboxes that combine different sound-producing mechanisms with extremely flexible and creative sequencers (Elektron, no date). The most prominent feature of Elektron's sequencers is parameter locking which allows any of the sound-shaping parameters to be 'locked' to specific values on a per-step basis. This feature enables the creation of sequences where any track's sound can change drastically from step to step, but more nuanced variations are, of course, also possible. All Elektron instruments have multiple tracks, and the sequencers on each track can have individual lengths and time scales. While the maximum length of a sequence is 64 steps, changing the time scale allows for the creation of even longer patterns than 4 bars at the expense of step resolution. Trigger conditions help to create even more variation by triggering steps based on probability or based on the loop count. The sequencers also contain a slew of shortcuts and other usability features improving the user experience, like copy-pasting multiple steps, recording movements of parameters to parameter locks in real-time, previewing individual steps, and muting steps. (Elektron, 2020b, 2020c, 2020a)

Figure 2.6. The Elektron Digitakt, an 8-track groovebox and sampler. Picture by Elektron used here with permission.

2.2.2. Pioneer SP-16

Compared to Elektron products, The Pioneer SP-16 offers a more user-friendly interface with a comparable feature-set but less advanced shortcuts. The device is a 16-track sampler and groovebox that uses stereo samples as its sound source. In contrast to most other grooveboxes, it features a touch screen in addition to its hardware controls for navigating and controlling the device. The sequencer is simpler than Elektron ones without any trigger conditions or advanced shortcuts but still featuring a virtually identical per-step parameter locking mechanism called ‘step modulation’ (Pioneer, 2019). Compared to Elektron devices, the SP-16’s user interface feels less cryptic, and the device is fast and intuitive to use. All functionality is readily available, yet some more advanced features, like trigger conditions, are a notable shortcoming. However, with the lack of features for effectively fine-tuning compositions, the SP-16 feels more focused on working on the bigger picture leading the user to different results compared to Elektron devices, which are arguably more geared towards finessing sounds and patterns.

Figure 2.7. The Pioneer SP-16, a 16-track groovebox and sampler.

2.2.3. Roland TR-8S

Roland's TR-8S is a device that combines a variety of synthesis methods with sample playback to produce sounds. It is marketed as a drum machine because it emulates sounds from Roland's classic drum machines, but it could also be classified as a groovebox since it is able to produce and sequence more than just percussive sounds. The device features eleven tracks, each track being either synthesis-based or sample-based with a freely assignable LFO and one selectable effect. A pattern contains a 16-step sequencer with eight variations, and each track's sequencer can have an independent length. Variations can be chained to create sequences longer than just 16 steps, which is important since no trigger conditions are available in the sequencer. The TR-8S offers no support for nudging triggers off the grid to create unquantised grooves. In contrast, micro-timing is available in both the SP-16 and in all Elektron devices. The TR-8S features a step modulation feature called 'motion record', which is more primitive than its counterpart in Elektron's sequencers. Only three parameters can be modulated at a time, two of them being fixed to the pitch and decay of the sound, while only the third one is freely selectable. However, it is also possible to specify accents, weak beats, sub-step trigger repeating, flams, and alternate sounds on a per-step basis to create further detail in a sequence. Compared to the grooveboxes examined in previous chapters, the TR-8S seems to be more geared towards performance use with its focus on tactile onboard controls and less on the screen. (Roland, 2020)

Figure 2.8. The Roland TR-8S drum machine.
Picture by Roland Corporation used here with permission.

2.3. Analysis

From the previous survey of existing applications, it is evident that browser-based grooveboxes are still far less advanced in their sound-producing mechanisms, sequencers, and user-interfaces than their hardware counterparts. As grooveboxes are mostly found in hardware form, the most innovation regarding grooveboxes resides in the hardware world as well. The apparent simplicity of current browser-based tools might be due to the novelty of audio-related web technologies and the isolated nature of browser-based tools, as interfacing with other applications located even on the same machine might be difficult. However, this isolation also applies to hardware grooveboxes to some extent: they are confined environments that allow only limited communication with the outside world. The similarity in terms of seclusion supports the choice of building a browser-based groovebox in particular since a groovebox can

produce entire compositions all by itself without having to interface with external tools. Taking inspiration from hardware devices is probably very useful for developing a new kind of browser-based application, and it also helps set reasonable constraints to keep the scope at bay and avoid creating an application that starts resembling a DAW.

Upon closer technical inspection of the previously presented browser-based tools, it turns out that none of the applications takes advantage of a particular modern web technology called WebAssembly. WebAssembly enables code to be executed at high speeds within the browser, which enables developing more ambitious applications than possible with conventional web technologies (WebAssembly Community Group, 2017). Selecting WebAssembly as one of the core technologies for the thesis project already gives it a tremendous theoretical advantage over existing applications. The employed technologies are discussed in more detail in chapter 3.3.

I have personal experience in using several hardware sequencers. In this work, inspiration is mostly taken from Elektron's product range as I am personally very keen on their approach to groovebox design and have a few of their products available for hands-on testing. The Elektron products most influencing this project are the Octatrack, the Digitakt and the Analog Rytm. The Octatrack and Digitakt are sample-based grooveboxes, whereas the Analog Rytm uses both analogue synthesis and samples to produce sound (Elektron, 2020a). Because the work here is based on a samples-only approach, the sound synthesis part is not of interest. Without the synthesis part, the Analog Rytm is almost identical to the Digitakt feature-wise. The Octatrack is also a powerful sampler and looper (Elektron, 2020c), but these features are not in the thesis's scope and are therefore left unexplored. Like mentioned previously, however, all of Elektron's sequencers are nearly identical.

3. Design and implementation

This chapter discusses the design and implementation details of the application. First, a couple of design principles are explained that act as the backbone for the later design decisions. The general overview of the application is then presented, followed by a description of the used technologies and the implemented sequencer features and sound processing functionality. Finally, the remaining user interface features are described, including shortcuts, randomisation features, macros, and the sample and pattern library.

3.1. Design principles

3.1.1. Encouraging serendipity

The first recorded usage of the word ‘serendipity’ was in 1754 by Horace Walpole. He derived it from a Persian fairy tale called “The Three Princes of Serendip”, where the protagonists made various accidental discoveries on their adventures (Zafarris, 2017). In more recent history, a similar concept has been famously used by painter Bob Ross (1987) in his television show “The Joy of Painting” with the term ‘happy accidents’. Both serendipity and happy accidents refer to the phenomenon of unplanned and accidental, yet fortunate, discovery. From personal experience, I have noticed that arriving at similar results in creative work by repeating similar types of choices can be detrimental to motivation and inspiration. In contrast, pleasantly surprising results are often achieved by accident. Because deliberate actions result in creations that are already familiar with the creator, they are more likely to be subjectively uninspiring or boring. It is vital that creative tools facilitate arriving at results that the user might not have arrived at intentionally.

There are several concrete ways to strive for serendipity in the context of a groovebox. An obvious way is to enable randomisation: by randomising a set of parameters, changes are made that might feel counterintuitive if made consciously, but such choices might still result in something exciting and unexpected. Another way to drive for happy accidents is to introduce macro parameters: particular types of parameters tied to multiple regular parameters in different ways. The more parameters are tied to a macro, the more unexpected the result of changing a macro parameter will be. Even providing eccentric effect processors, as the frequency shifter described later in this chapter, can help in coming up with unanticipated sonic output.

3.1.2. Ease of use and virtuosity

When trying to offer much complex functionality in a software application, it quickly leads to cluttering the user interface with controls, menus and other options. Especially when wanting to provide versatile features, the balance of providing profound control while maintaining an easily usable interface is very delicate. However, I believe that such a balance is worthy of being pursued, even if it could not be perfectly achieved in the end. This sentiment is echoed by Wessel and Wright (2002): “We believe that ’getting started’ with a computer-based instrument should be relatively easy, but this early stage ease-of-use should not stand in the way of the continued development of musical expressivity. … Is a low ‘entry fee’ with no ceiling on virtuosity an impossible dream? We think not and argue that a high degree of control intimacy can be attained...”. A closely related design principle for computer music controllers is “instant music, subtlety later” (Cook, 2001). First-time users should be able to produce sound and get a grasp of the basic functionality very quickly. In contrast, long-time users should be able to discover advanced functions that only become evident after prolonged use. Most current browser-based grooveboxes still only provide toy-like interfaces that the user quickly outgrows, so almost any improvement on the workflow will already be noticeable in comparison.

An example of a product that is relatively easy to use from the start while offering plenty of features for advanced users is Elektron’s Analog Rytm, which is a hardware drum machine and conceptually close to the application developed here. Pressing one of the 12 pads selects a track and pressing on the sequencer’s buttons immediately inputs triggers, so creating basic rhythms is trivial without opening the manual. A knowledgeable user will also quickly understand how to edit sound parameters by using the encoders on different pages. Many more in-depth functions can either be discovered by accident or be learned by referring to the over 100-page manual where a plethora of shortcuts and power-user key combinations are explained in detail. The machine will reveal new functions even to the most experienced user still after extensive periods of active use. Although the feature scope of the Analog Rytm is much larger than that of this project, it still serves as an apt example of the paradigm.

3.1.3. The importance of the user interface

Especially when building a creative tool, it is not sufficient for it to only fulfil its theoretical function, which in this case is producing interesting musical compositions. As an extreme example, it would be theoretically possible to achieve any sound imaginable with a system that allows the user to set the digital output signal’s individual sample values. However, it is obvious why such an interface would not be usable. Systems that allow for near limitless possibilities can paralyse the creator, turning them

into more of an engineer than a musician (Magnusson, 2010). If the tool feels laborious or uninspiring to use, the user will less likely utilise the tools theoretical functions to their full potential. Predictability is also an important trait; users should always feel that they control what they are doing (Wessel and Wright, 2002). The user experience design should ensure that every action has a logical consequence and minimise the number of actions required from the user to accomplish specific tasks. Visual interfaces should be designed to give distinct clues to the user about the expected behaviour of different controls. For example, colour-coding related components, applying skeuomorphism where appropriate, and animating key movements, all help the user to create a useful mental model of the tool. However fantastic a tool might be in its technical potential, the user interface will always be the ultimate translator of human input, dictating how a tool will be used. Although the sound engine, of course, also constrains the creative possibilities, the user interface ultimately acts as the mapping engine turning user input into commands for the sound engine defining the applications' actual functionality and constraints (Magnusson, 2010).

3.1.4. Audio quality

A tool designed for musical work has to have a high enough standard in terms of its audio quality. For example, some applications might require that audio playback systems are alias-free when repitching audio by resampling it. However, in this project, aliasing caused by resampling audio might not only be a negligible effect but even a desired quality. Producing alias-free resampling does not come free as it requires oversampling the audio and using a low-pass filter, which is why some classic hardware samplers limited by available processing power introduced significant aliasing in the process, which can be regarded as an aesthetically pleasant sound (Aisher, 2015). On the other hand, designing envelopes to avoid a clicking sound when their curve changes abruptly is crucial as it can be regarded as an unpleasant sound. The required quality of different audio processors is derived by comparing them loosely to similar ones in hardware grooveboxes.

3.2. General architecture

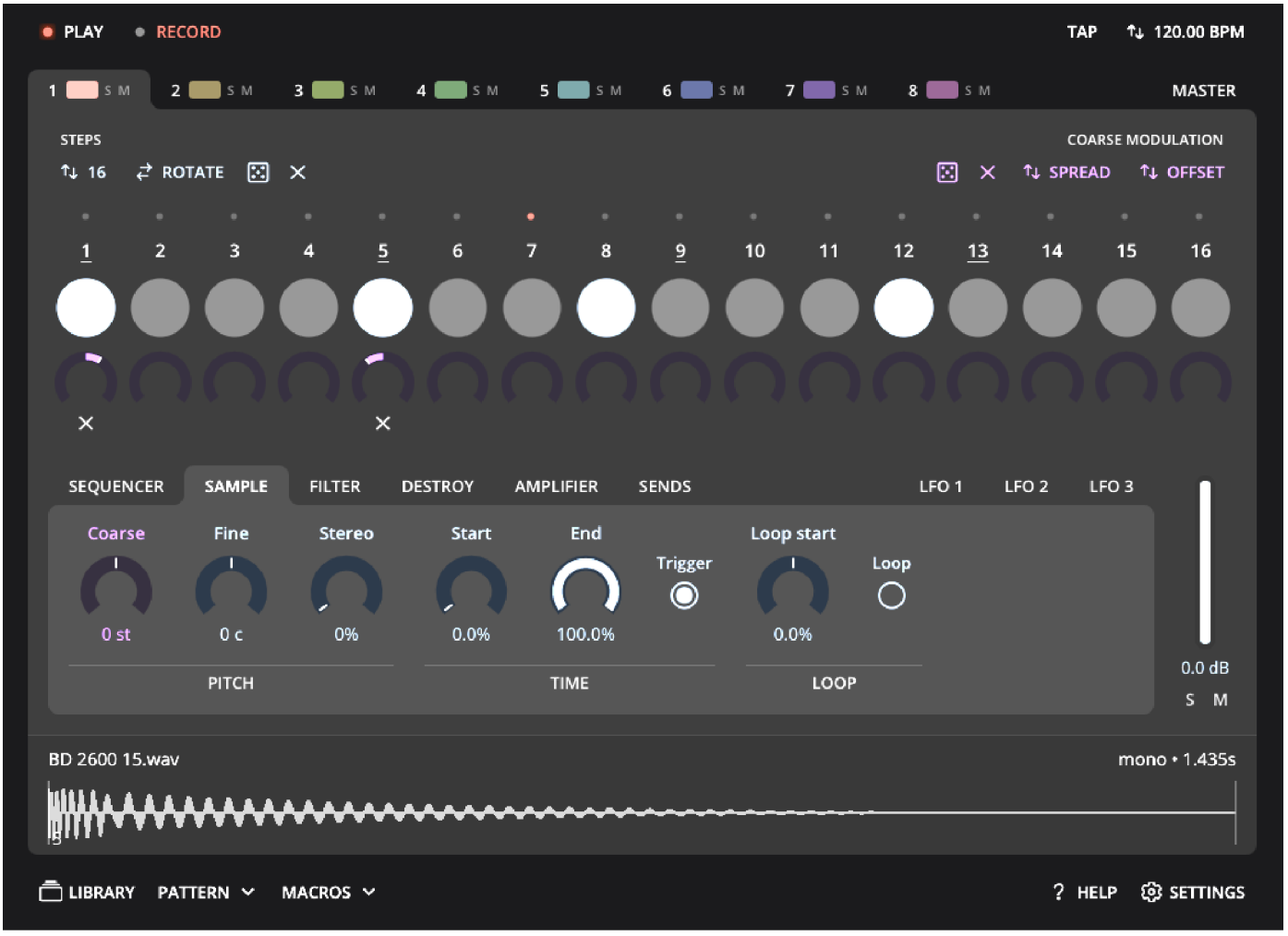

Figure 3.1. Screenshot of Tahti's interface

The application takes advantage of a tab-based layout to organise its controls. The top-most numbered tabs allow switching between the eight tracks and the master track. Only the sequencer and sound-shaping parameters of the currently selected track are displayed at a time. Each tab has a signal indicator with a unique colour to quickly identify what kind of sound is produced on each track. On the 16-step sequencer, the lighter circles represent the active steps. The control below each circle is used to set the per-step modulation for the currently selected parameter. Steps on which modulation for the selected parameter is active have a cross icon below the modulation control which allows removing the modulation from that step. Above the sequencer are controls for changing the sequence length along with macro controls and randomisation features. Below the sequencer, the track-specific sound-shaping parameters are grouped into another tabbed container. A light red colour indicates the currently selected parameter, and the selection changes whenever a control is clicked. As mentioned previously, the modulation row in the sequencer controls modulation for

the currently selected parameter. On the right side of the sound-shaping parameters is an output level slider for the current track along with solo and mute buttons. The sample of the currently selected track is visualised at the bottom as a waveform with start and end position indicators layered on top of it per the track's start and end parameter values.

Outside of the track-specific container are a few global functions. On the top, playback can be toggled along with the record mode, which, when engaged, allows for live movements of parameters to be recorded as step modulation. The tempo can be changed either via the numeric control or by repeatedly clicking on the 'Tap'-button. On the bottom, the sample and pattern library can be accessed via the 'Library'-button. Samples and patterns can be imported into the library from a user's computer and organised into folders much like on a real file system. The 'Pattern'-dropdown menu contains options for saving the current pattern, creating a new pattern, downloading the pattern as a file and rendering the pattern to an audio file. The 'Macros'-dropdown menu houses options for quickly resetting all sequencers or parameters for either the current track or all tracks at once. On the right side, the 'Help'-button opens a window containing explanations of the general concepts, and the 'Settings' button opens a window for configuring a few settings. The settings allow configuring the overall latency and whether to display interactive help texts, which are explained more in detail in chapter 3.6.

3.3. Utilised technologies

The application consists of the audio engine and user interface, both of which employ different technologies. The audio engine is written in a novel open-source programming language called SOUL (SOUnd Language), which was selected due to its promises regarding high performance and ease-of-development (ROLI, 2020a). Although a stable version of SOUL has since been released, the language was still in beta development when starting the thesis, and I found and reported numerous bugs during the project. Luckily these were quickly acknowledged and fixed by the SOUL development team.

WebAssembly is a crucial modern web technology required by the application as it allows binary code to be executed within the browser with near-native performance (WebAssembly Community Group, 2017). The SOUL toolset supports compiling SOUL code as C++ or directly as WebAssembly code (ROLI, 2020b). This code, however, only implements a very basic interface for data flow between the browser and the code, supporting merely simple parameter data to be passed to the SOUL application (ROLI, 2020c). As more complex bi-directional data flow is required here, an additional communication layer had to be devised in C++, wrapping the compiled SOUL C++

code with a custom communication interface. The wrapped code is then compiled to a WebAssembly module using the Emscripten toolset. Another critical web technology used here is the AudioWorkletNode, which enables running the WebAssembly module on a dedicated audio thread to avoid the main thread from interfering with the audio processing (W3C, 2021).

The user interface is written using more conventional web technologies, including HTML, JavaScript and CSS. I have a professional background in front-end web development and have tried many different libraries and frameworks during my career. I chose React.js as the primary user interface library because of my positive personal experiences with it. Much like SOUL's promises of easy development, I feel like the same applies to React as well: it encourages breaking up the code into smaller chunks for maintainability and code reuse. The code itself also features several developer-friendly paradigms that make development more pleasant.

3.4. Sequencer

Although most hardware grooveboxes feature sequencers with at least 64-steps, only a 16-step sequencer is implemented here to keep the user interface more straightforward and not take up as much screen real-estate. To compensate for this limitation, many sequencer features are implemented to allow for variation beyond only 16-step loops.

First, the length of a sequence can be adjusted between 1 and 16 steps, allowing for creating polymetric patterns where sequences of different lengths can be played simultaneously while still maintaining the relative duration of a step.

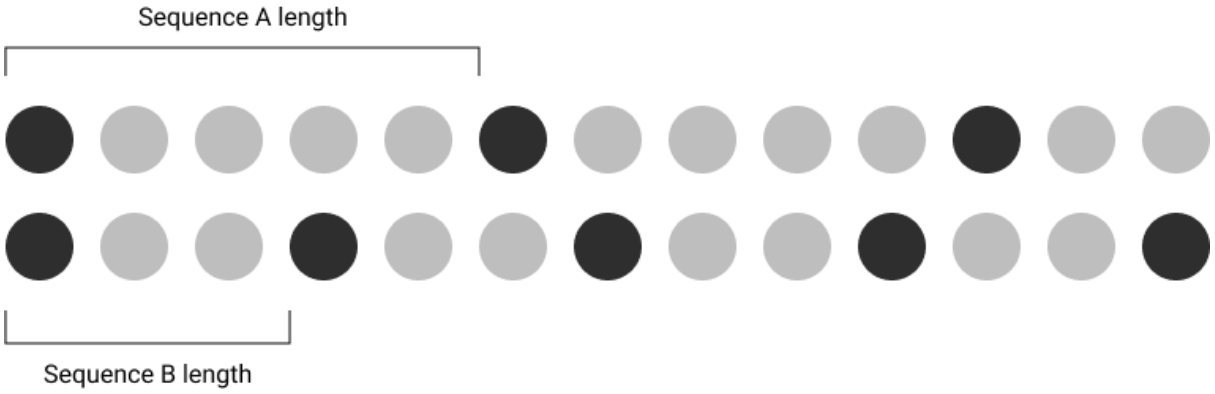

Figure 3.2. A 5:3 polymetric pattern
The length of two sequences differs while the relative duration of a step remains the same.

Control over the speed of a sequence can also be adjusted with two notable benefits: the creation of polyrhythmic patterns when used in addition to the sequence length control and the creation of patterns that vary over longer periods than just one bar at the regular speed.

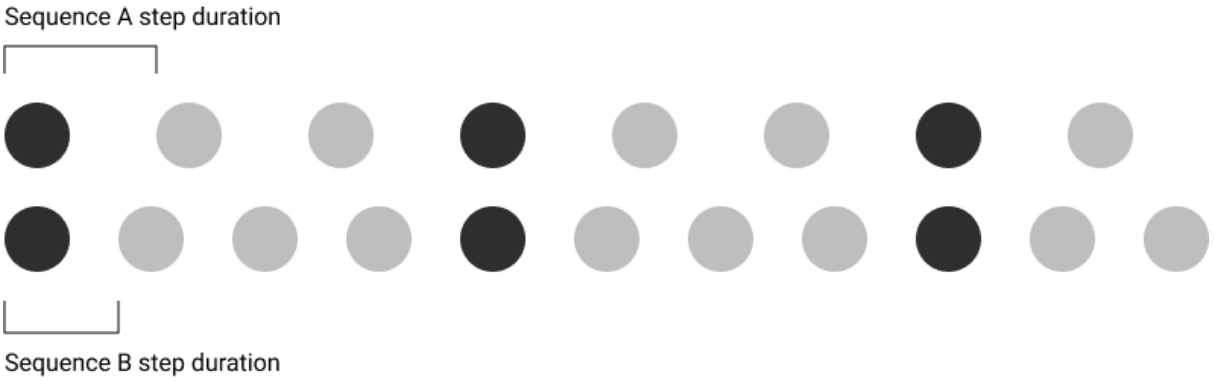

Figure 3.3. A 3:4 polyrhythmic pattern
In a polyrhythmic pattern, sequences might have the same duration while the step durations (speed of the sequences) differ.

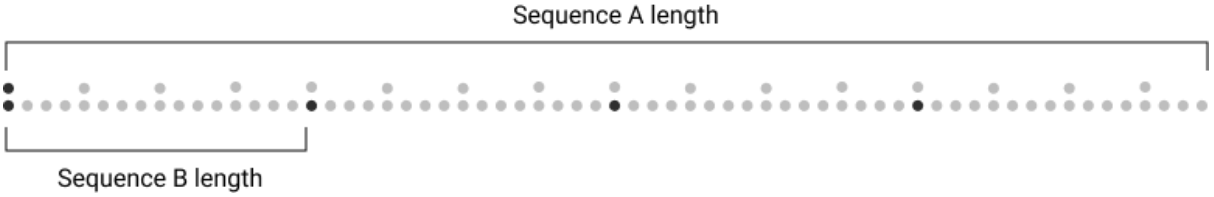

Figure 3.4. Increasing temporal variation by sacrificing step resolution
Here, sequence A runs at a 25% speed compared to sequence B. Both sequences still only have 16 steps but sequence A can contain variations over a longer period by sacrificing the resolution of steps.

Elektron sequencers offer control over pattern length, allowing for polymetric rhythms just like discussed above. However, their sequence speed option only has a couple of preset options inhibiting the creation of arbitrary polyrhythmic patterns. In this sense, the sequencer presented here already provides more possibilities to experiment with. To make things even more interesting, the pattern speed parameter can be modulated on a per-step basis, making the pattern advance faster or slower on a certain step while changing to a different speed on another step.

Trigger conditions are an essential tool for avoiding repetitions of a sequence each iteration. In Elektron sequencers, different trigger conditions are mapped on one single parameter, starting from conditions that allow the step to trigger based on whether the previous step was triggered followed by probability-based triggering and triggering

based on the loop counter (Elektron, 2020b). Taking inspiration from Elektron-style trigger conditions, three parameters are implemented: probability, wake up and snooze. Probability determines how likely it is that a step will be triggered, wake up determines how many times the sequence must loop before a step starts triggering, and snooze determines how many loops it will take for a step to trigger again after it has triggered. Separating the conditional trigger parameters allows using the probability condition in combination with loop-counting conditions, which permits interesting variations to patterns not possible even on Elektron sequencers. Conditions based on previous steps are not implemented but would be an easy addition if they are deemed necessary in the future.

A slew of functionality affecting the step timing is also present. Firstly, control over the offset of step timing is provided to allow breaking out of the quantisation level of the steps, making more unquantised ‘humanised’ grooves possible. A toggle parameter controls whether the step starts at the specified time, like how step-sequencers typically operate, or if the sample should be played so that it ends at the start position. Such a playback mode is useful when playing audio in reverse and wanting to sync the end time with the start of another sound and is a novel feature that I have not found on any other sequencer.

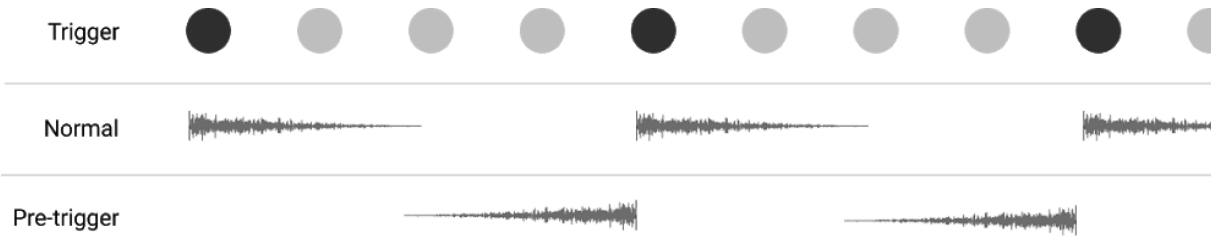

Figure 3.5. How sample-playback is affected by the pre-trigger toggle parameter.

Taking further inspiration from existing groovebox sequencers, retrigger or trigger repeat functionality is implemented, which allows repeating a step a controllable amount of times with a controllable interval. The interval is presented in steps, so it is easy to create rhythmic repetitions that are either shorter than a step for trills and flams or longer than a step for triplets, for example.

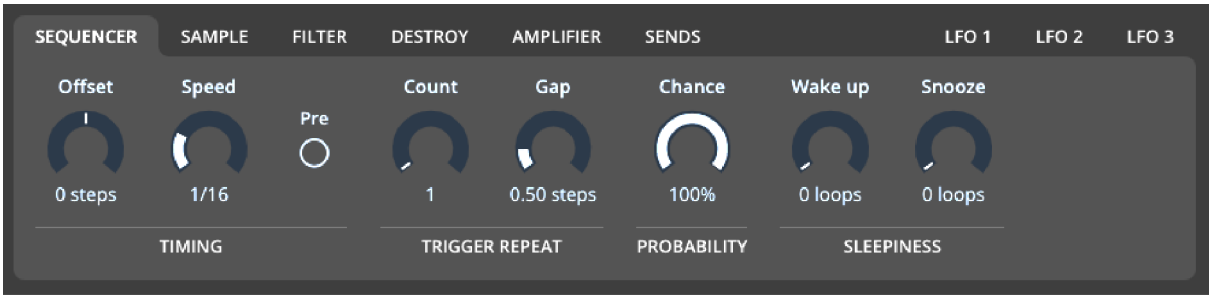

Figure 3.6. Sequencer-related parameters.

Being able to modulate all of the sequencer's parameters on a per-step basis opens up many different possibilities and a lot of room for creative experimentation. With the help of conditional parameters, it is effortless to break free from repetitive 16-step patterns while the actual number of steps still remains at only 16. With the different time-based options, the timing of sounds can be programmed to have a great amount of nuance. This results in a feature-rich sequencer which is the backbone of a self-contained groovebox.

3.4.1. Step-based modulation

Step-based modulation can be implemented in ways where modulation values are either absolute or relative in relation to the global parameter values. In a relative modulation scheme, modulation is represented as an offset from the global parameter value. By default, every parameter has a modulation value of zero for each step, and changes to the modulation are summed to the parameter's current value. This way, when changing a parameter, step-modulated values will also change relatively. For example, if the parameter's value is set to 70% globally and the step has a modulation value of +19% for that parameter, the parameter will be at 70% + 19% = 89% during that step. On the other hand, absolute modulation allows each parameter to be locked to a specific, absolute value on each step. This means that changes to the global value of a parameter will not affect steps where the parameter value has been locked to a different value. For example, with an absolute modulation scheme, if a user has specified that a step has a value of 89% for a certain parameter, changes to that parameter will have no effect during this specific step. The parameter will only affect the sound when the sequencer is at a step where the parameter is still in an unlocked state.

Both modulation schemes have distinct benefits and drawbacks. With relative modulation, changes to global parameters will always result in some change in the sound. In contrast, with absolute modulation, a particular global parameter might be rendered entirely useless when all of the steps have a modulated value for that parameter. Maintaining carefully programmed modulation values is more reliable in an absolute modulation scheme, however, as accidental changes to global parameters in a

relative scheme might completely change carefully selected modulation values. Relative modulation schemes are problematic when wanting to record parameter changes live by moving a control: when the global control for a parameter is moved, the value to which modulation is relative to is constantly changing. This could be worked around by storing the parameter's value when starting to record and restoring it to the stored value afterwards, but at the cost of technical and user-facing complexity. In a sense, absolute modulation might also be a more explicit paradigm for the user because the values that are displayed per-step are precisely what the user will experience. In contrast, relative modulation values will always be summed to the global value of a parameter. Absolute modulation is also implemented in various hardware grooveboxes, like in products developed by Elektron, Roland, Korg and Akai. In fact, there does not seem to be any hardware groovebox that would use a relative modulation scheme at the time of writing. (Korg, 2014; Roland, 2019; Akai, 2020; Elektron, 2020c, 2020b, 2020a; Polyend, 2020)

As is evident from the previous paragraphs, it is not immediately clear whether one modulation paradigm is superior to the other. For the work in this thesis, however, the absolute modulation scheme is selected, mainly because the technical implementation is considered to be easier and because it might also be more comprehensible for end-users. Besides, for users with previous experience with hardware grooveboxes, the absolute modulation scheme will be instantly familiar.

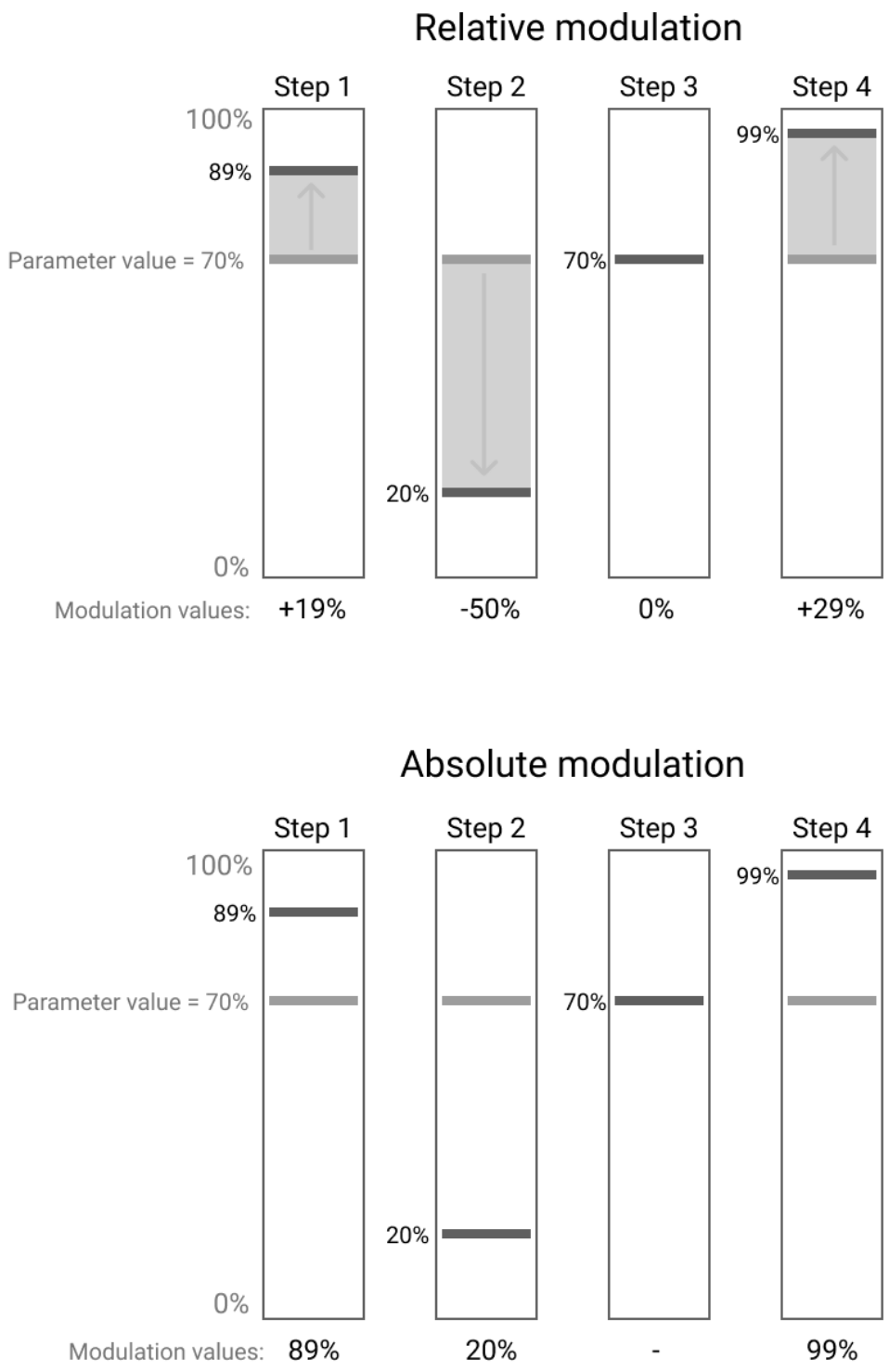

Figure 3.7. Visualising the difference between absolute and relative modulation. In a relative scheme, the modulation values are added to the parameter value (70%). In an absolute scheme, the modulation values override the parameter value.

3.5. Sound generation and processing

3.5.1. Sample playback

The sample playback algorithm determines the different ways in which the track's sample data can be used. Each track has monophonic stereo sample playback, meaning that multiple tones, or notes, can not overlap. Monophonic playback is standard and well suited for grooveboxes as they are mostly designed for composing rhythmic content instead of polyphonic melodies. However, monophonic melodies are entirely possible to create because the pitch of the sample can be controlled. Changing the pitch is implemented by resampling the audio in real-time. The start and end position of the sample can be specified, and the end position can be placed before the start position, in which case the sample playback will be reversed; the playback will always move from the specified start position towards the end position. The sample playback can also be looped indefinitely, useful, for example, in combination with single-cycle waveforms as the sound source, essentially turning the track into a monophonic static synthesiser voice. The pitch can be offset in both the left and right channels to help widen mono samples in the stereo image. This function is inspired by a similar control in Ableton's Simpler sampler (Ableton, 2020a). The sample playback is not oversampled, so some aliasing occurs when pitching up sounds with higher frequency content. This could be trivially fixed in a future version, but for now, I am considering it a desired effect as it is a sound reminiscent of old hardware samplers as discussed in chapter 3.1.4.

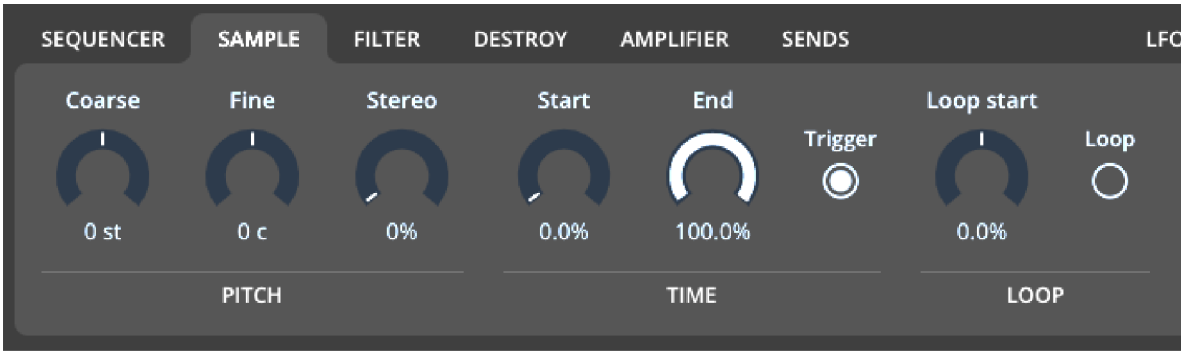

Figure 3.8. Screenshot of the sample playback parameters.

3.5.2. Filters

Filters are an essential part of any groovebox because of their ability to sculpt the sound's frequency contents. Elektron grooveboxes feature a filter architecture where a low-pass and high-pass filter are chained in series, which makes it possible to achieve a band-passed frequency response in addition to low-passed and high-passed responses (Elektron, 2020b). The Pioneer SP-16's software filter offers the choice between a low-pass, high-pass and band-pass filter, and between 12 dB and 24 dB per octave slopes (Pioneer, 2019).

For this application, a multi-mode state-variable design is selected, which allows morphing between low-pass, band-pass and high-pass frequency responses with one control. The filter design found in Elektron's machines has the drawback that an additional parameter has to be included for specifying which of the two frequencies the filter envelope affects. With the mode-control, such complexity can be avoided as there is only one cutoff frequency to modulate while still offering all three types of filter responses. The drawback is that the width of the band-passed response can not be adjusted, but this is considered a minor issue. While this architecture currently does not allow specifying the slope of the filter cutoff, such functionality would be trivial to add at a later time, but it was not deemed necessary at this point.

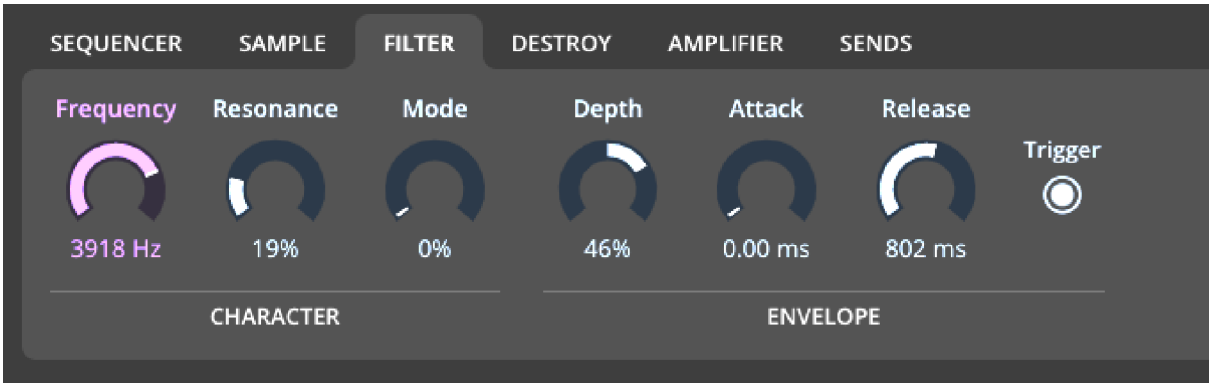

Figure 3.9. Screenshot of the filter parameters.

In addition to per-track filters, filters are also used in the implementations of other processors. The entire distortion signal chain is succeeded by a one-pole filter that can either be a low-pass or a high-pass one. The reverb, delay and phaser signal chains also incorporate high-pass and low-pass filters. One-pole designs are selected for these contexts as they offer the most gentle 6dB per octave attenuation and are slightly more efficient computationally than steeper filters. (Redmon, 2012)

The standard filter library of the SOUL language luckily contains an implementation of a resonant multi-mode SVF filter and one-pole filters that are suitable for modulation and could directly be implemented for use in this application (ROLI, 2021). Modulatability is an obvious requirement for the filters as parameter values should be able to change on a per-step basis and be modulated with LFOs. The implementations in SOUL are derived from established works by Pirkle (2013) and Zavalishin (2015).

3.5.3. Distortion

The distortion section consists of several processors that distort and mangle the sound in various ways. First in the chain is a waveshaper which is useful for introducing saturation, especially for sounds with lower frequency content. Next, the signal is fed through a frequency shifter which also allows offsetting the amount of the shift between the left and right channels for a wider stereo image. Frequency shifting is an unusual yet interesting effect, especially for inharmonic sounds, or the creation of such since it shifts all frequencies in a sound by the same absolute amount disassociating its harmonic relationships (Clark, 2003). Because the technical implementation of a frequency shifter also easily enables ring modulation, a ‘Ring’-toggle is added for turning the effect into a ring modulator. When the ‘Clean’-toggle is engaged, a high-pass filter is added to the signal chain to remove negative frequencies wrapping back into the audible range when negative frequency shifts are applied.

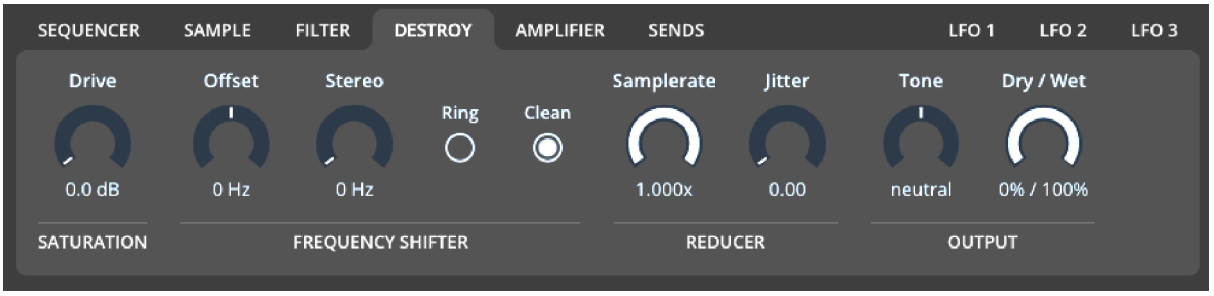

Figure 3.10. Screenshot of the distortion parameters.

After the frequency shifter, the signal is fed to a sample-rate reducer. Sample-rate reduction is a digital-sounding effect producing harsh high-end frequency content and is found on many hardware grooveboxes (Pioneer, 2019; Elektron, 2020c, 2020a, 2020c; Roland, 2020). The ‘Jitter’ control can be used to soften the harshness of the effect or to produce subtle noise even when the sample-rate control is at the maximum position. Finally, the signal goes through a high-pass or low-pass filter as controlled by the ‘Tone’ control, and the distorted signal can be mixed in with the original input signal with the ‘Dry / Wet’ control. Low-passing might be useful when wanting to reduce the harshness produced by some of the processors, and high-passing might be useful when, for example, only wanting to mix in the higher frequencies of the distortion chain.

3.5.4. Envelopes

Numerous flavours of envelopes exist, but certain types are more suitable for grooveboxes than others. As the sequencer triggers are exempt from information about an event's end time, the attack-hold-release (AHR) (also known as attack-sustain-release) envelope type was selected as the main envelope. An AHR envelope can be controlled with three parameters, one for each stage of the envelope.

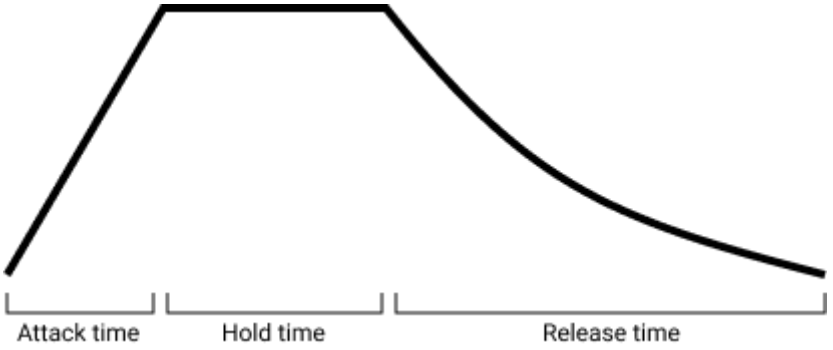

Figure 3.11. Visualisation of an AHR envelope.

A linearly rising attack time is typical in many applications, which is also deemed sufficient here. The shape of the release curve is much more important for percussive sounds, however. To aid in discovering an ideal shape, the release curves in the AHR amplification envelopes were examined closely in three existing products: the Analog Rytm, the Pioneer SP-16, and the Ableton Simpler software sampler. A sine wave was used for each test, and the release times were adjusted to arrive at curves with roughly similar lengths.

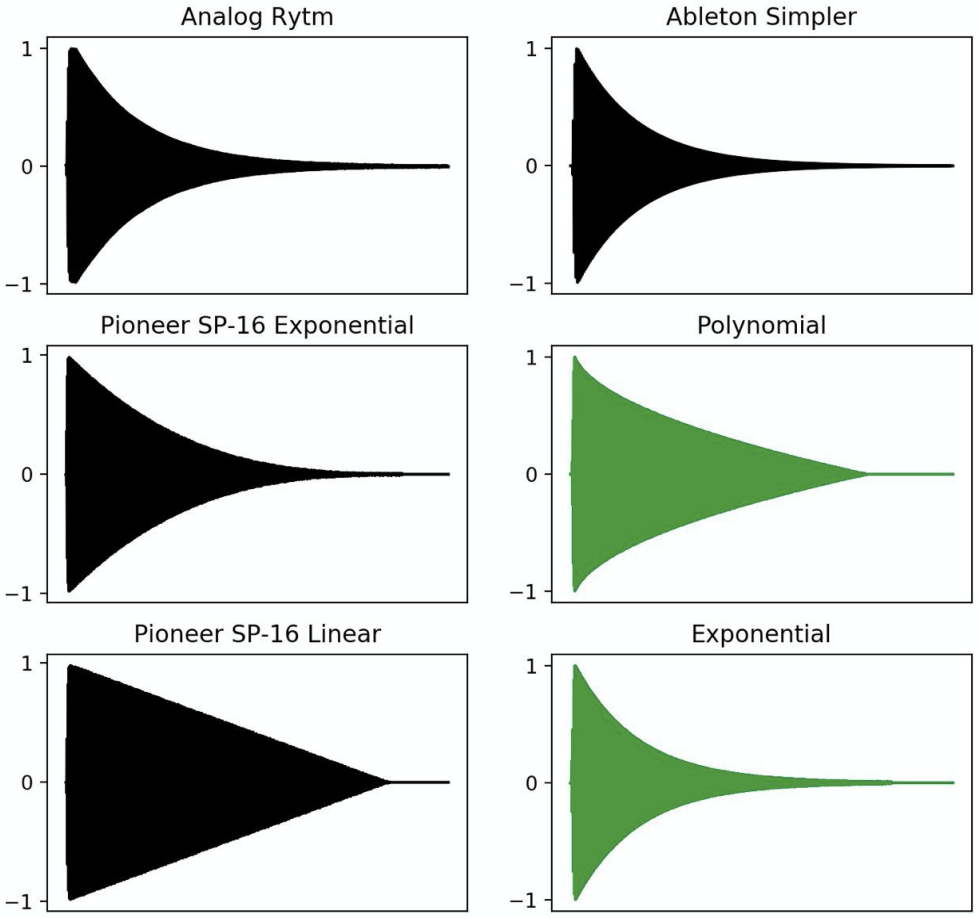

Figure 3.12. Measured amplification envelopes. Black curves are waveforms measured from the labelled devices, while green curves are the ones tested during development.

The Analog Rytm and Ableton Simpler do not allow the user to adjust the shape of the release curve; both have an exponential shape by default. The SP-16 allows switching between linear and exponential modes for the release shape, linear curiously being the default here. The exponential shapes of the Analog Rytm and Ableton Simpler seem almost identical, and the SP-16 in exponential mode is close as well. However, it does not seem to decay quite as quickly in the beginning. Compared to a linear decay, the exponential shape is found to be more suitable sonically; it seems to result in a snappier start and a more gradual fade-out. In comparison, the linear shape sounds like it goes silent abruptly. A polynomial shape is also experimented with, but it has similar problems to the linear one, although the sound's beginning is slightly more pronounced when controlling amplitude. The final implemented exponential shape is displayed in the lower right corner in Figure 3.12. and labelled as 'Exponential'.

The downside of an exponential shape is that mapping the release time parameter value to a meaningful value in seconds is vaguer as theoretically, the envelope decays infinitely. However, this is considered a minor problem as the sound is more important than displaying relevant units to the end-user. In addition to being used for modulating the amplitude, the developed envelope is also used as the dedicated filter envelope.

3.5.5. Low-frequency oscillators

Low-frequency oscillators (LFOs) are essential modulation tools and are only very rarely exempt from grooveboxes or synthesisers in general. The basic parameters found on virtually every LFO are shape, speed and amount, which defines the minimum and maximum values between which the oscillation moves. The most basic shapes are sine, sawtooth and square waves, while the modulation speed is usually between 0 and 20 hertz, as oscillations faster than 20 hertz start to become audible as frequencies.

To make the LFOs work for a broader range of use-cases, additional parameters can be devised. A reset toggle determines whether the LFO starts from the beginning of the oscillation cycle when it receives a trigger from an external source, like the sequencer in this case. Adding a reset parameter creates the urge for a phase parameter, which determines at which point of the oscillation the LFO will start at when retriggered. This is usually expressed in degrees, where a 180-degree phase would mean starting at the middle of the shape. A sample-and-hold toggle can be added to ‘hold’ the output of the LFO whenever an external trigger is received. Adding random noise to the shapes now makes much sense since combining noise with sample-and-hold will turn the LFO into a random value generator, outputting a random value each time a trigger is received. Specific LFO shapes, like the saw or an exponential decay, function well as simple envelope generators when the LFO has a toggle-parameter that controls whether the shape is looped or played only once when triggered. The LFOs of Elektron grooveboxes feature many of the functions listed in this chapter, and many are also common in other grooveboxes.

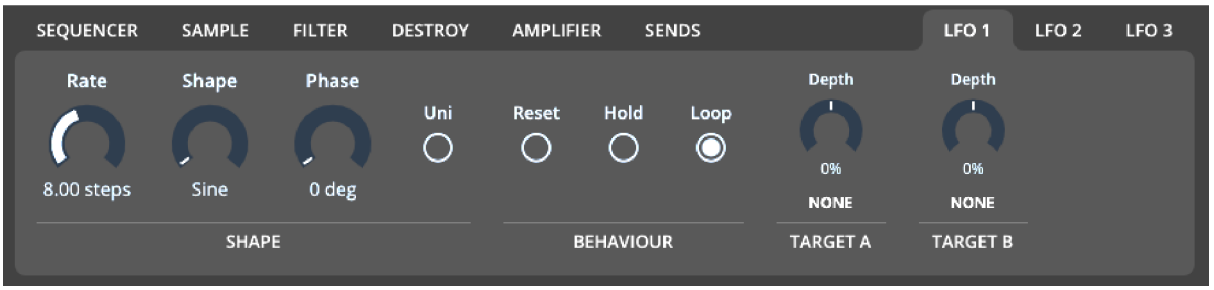

Figure 3.13. Screenshot of the LFO parameters, identical to each of the three LFOs.

For this application, three LFOs per track are implemented. All of the LFOs are identical, sharing all of the features explained in the previous paragraph. The implemented shapes are sine, triangle, saw, square, exponential decay, and noise. All of the shapes are bipolar by default, meaning that they range from -100% to +100%, but another parameter toggle is added to make them unipolar, changing their range from 0% to 100%. The LFOs have two freely assignable destinations with dedicated depth controls that range from -100% to 100%.

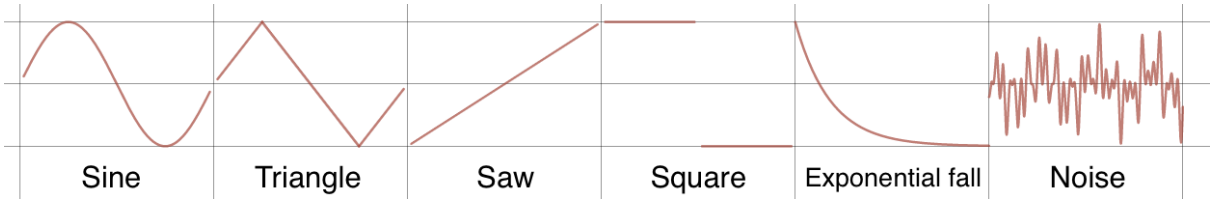

Figure 3.14. Visualisation of the different LFO shapes

By combining all the shapes, the polarity toggle, the start phase parameter, and other options, a range of modulation options can be achieved, including classic LFO behaviour, various simple envelopes, and random value generators. The unit of the rate-parameter is measured in unquantised steps, so it is easy to achieve both tempo-synced and freely moving modulation. LFO's can modulate each other to create even more interesting and unpredictable movement. Like any other sound-shaping parameters, also all of the LFO's parameters can be step-modulated for even more variation.

3.5.6. Global send effects

Global send effects are shared among all of the tracks in a groovebox. Send effects only have one set of parameters, and the audio output of each track can be sent to each global send effect by a controllable amount. Common choices for send effects in grooveboxes include computationally heavy effects like reverbs and effects that could be used to glue sounds together.

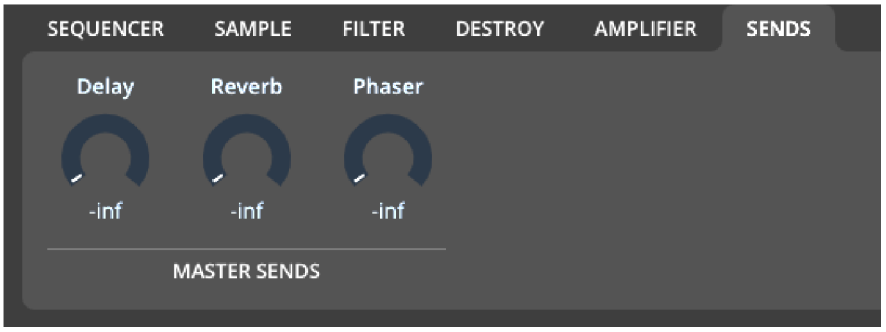

Figure 3.15. Screenshot of the per-track send controls.

Reverb is an integral part of a groovebox for giving the sounds a sense of dimension and space; one exists in virtually every groovebox on the market. Unfortunately, it is difficult to analyse the architecture of existing reverbs solely based on their output, so the implementation has to rely on readily available information. The implemented algorithm is based on the late reverberation network presented in an article by Anderson and Costello (2009). It consists of four parallel all-pass filter chains, each containing three all-pass filters tuned not to resonate with each other. Parameters allow for control over the decay time, the frequencies of both a low-pass and high-pass prefilter, the stereo width, the amount of modulation, and the pre-delay duration in milliseconds. Only one reverb is available as a master send because the algorithm is more CPU hungry than most other processors in the application. This is probably why reverbs are usually found as send-effects on hardware grooveboxes (Pioneer, 2019; Elektron, 2020b, 2020a; Roland, 2020).

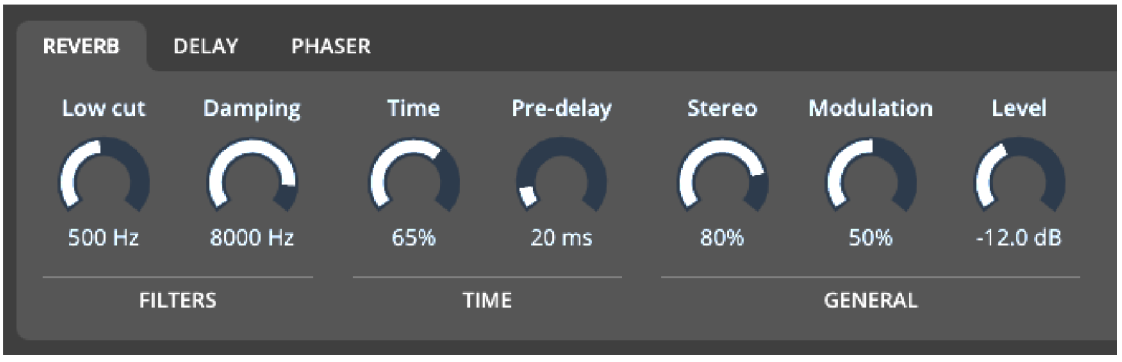

Figure 3.16. Screenshot of the reverb parameters.

The phaser is based on the simplified phaser design presented by Pirkle (Pirkle, 2019), where the input signal is fed through a series of all-pass filters, the frequencies of which are modulated by a single LFO. The amount of all-pass filters can be adjusted from two to eight. The phaser features a feedback path with an adjustable gain running from its filters' outputs back to their input to add even more flavour to the effect. For controlling the effect's character, the spread of the all-pass filter frequencies can be adjusted, and a separate parameter for adjusting the stereo offset of the frequencies is available. Since a phaser can accumulate a lot of gain in the low-end when used with bassy sound sources, and because the phaser here is a global send-effect, a high-pass filter is added to the beginning of the signal chain to account for cases where too much low-end might accumulate. The implementation differs from common phasers in that there is no built-in LFO for modulating the centre frequency. Instead, one of the three master LFOs should be used when a modulated phaser sound is desired. This design decision allows the use of unusual LFO shapes in addition to the sine and triangle shapes that are usually hardcoded into phasers, potentially resulting in more interesting results.

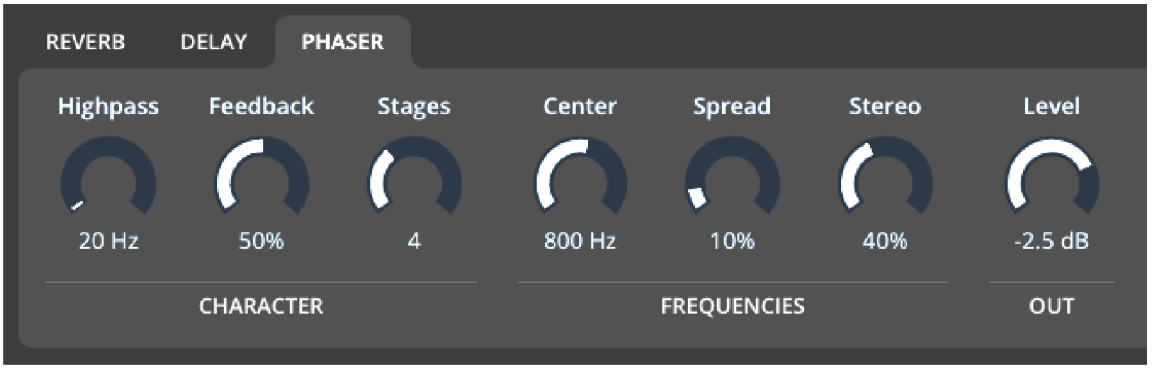

Figure 3.17. Screenshot of the phaser parameters.

Delays are useful for creating rhythmic layers and a different sense of space than reverbs. A rudimentary but comprehensive set of parameters are implemented for the delay here: cutoff frequencies for the high-pass and low-pass filters, a delay time parameter, a stereo spread parameter, and a feedback parameter. The time is represented in unquantised steps, so tempo-synced delays are possible in addition to off-beat timings. The time parameter is also interpolated, which means that changing it will not result in clicks in the sound but a re-pitching effect reminiscent of tape delays. This unlocks creative uses when modulating the delay time with an LFO. The filters are placed into the feedback path instead of in front of the signal chain to support decaying the frequency content of the sound in addition to the level. The delay is implemented as a send to save on performance and reduce per-track complexity.

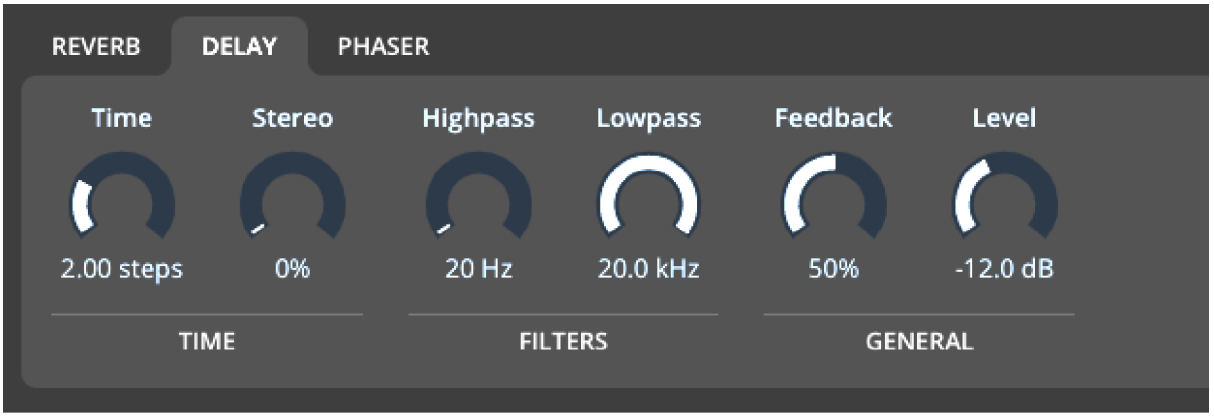

Figure 3.18. Screenshot of the delay parameters.

3.6. Facilitating learning

The application features an interactive help text to aid in flattening the learning curve for first-time users. The text is visible on the bottom of the page and changes according to whatever is beneath the mouse cursor. Texts explain the user interface in general, but also describe each parameter. The feature is inspired by a similar one in Ableton Live (Ableton, 2020a). Interactive help texts are useful for a quick reference on specific functionality, but an additional 'Help' window is also available, explaining the general concepts of the application and listing all available shortcuts.

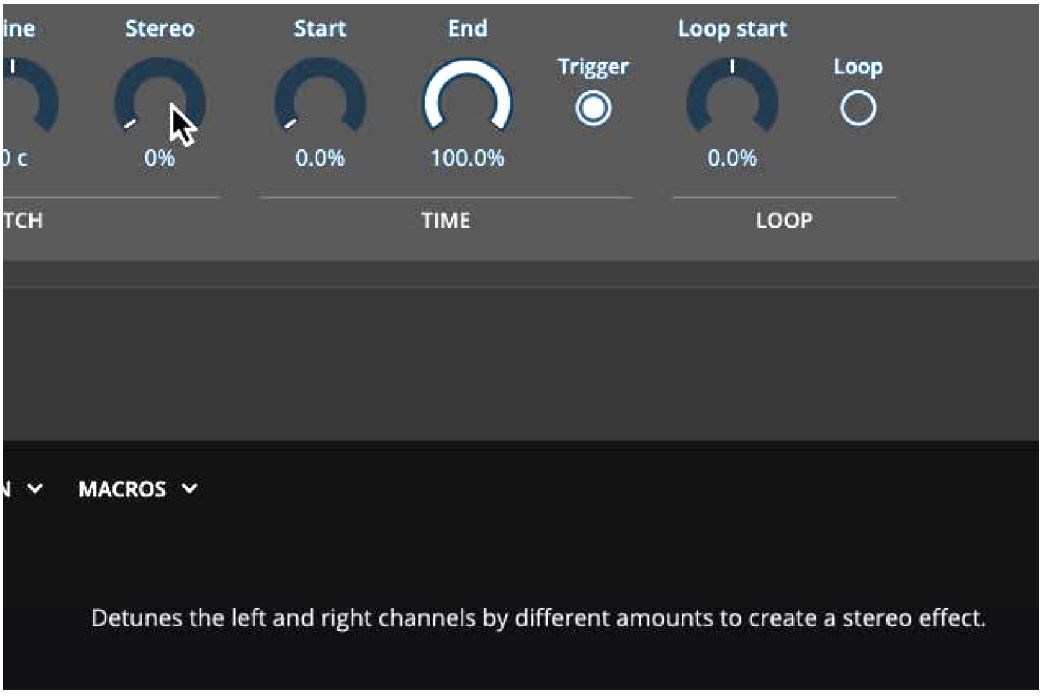

Figure 3.19. Hovering the cursor over a parameter shows a description of that parameter at the bottom of the page.

3.7. Shortcuts

One reason why hardware grooveboxes are often so enjoyable to use is that they provide tactile, quick control via their interfaces by providing key combinations and other shortcuts for standard functions. A similar level of usability can be tried to achieve for a browser-based groovebox for the desktop environment by implementing a set of keyboard shortcuts that work in tandem with the mouse.

All of the controls related to some numeric value work with the same usability shortcuts. Double-clicking on a control will return the related parameter to its default value while dragging a control while holding down the shift key will change the value more slowly for fine-tuning it. After focusing a control by clicking on it, the keyboard can be used to write in an exact value manually with the help of a little display that will appear.

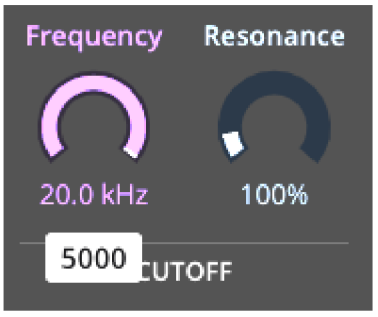

Figure 3.20. The frequency is changed to precisely 5000 Hz by clicking on the 'Frequency' control, writing "5000" on the keyboard, and pressing the enter key.

A set of shortcuts is available for working with steps and modulation on the sequencer. Steps can be copy-pasted by hovering over a step with the mouse and pressing the C key to copy and the V key to paste while hovering the cursor on another step. All modulation can be erased from a step by similarly hovering over it with the cursor and pressing the X key. In addition to copying steps, sometimes it is also useful to move a step to the left or right on the sequencer. This can be achieved by hovering on a step and pressing either the left or right arrow keys. The same could also be achieved by copying, pasting and deleting the initial step but the manoeuvre is deemed to be common enough for it to warrant its dedicated shortcut.

The hardware Cirklon-sequencer has a unique feature for creating linearly increasing or decreasing sequences of step modulation called slope edit (Sequentix Music Systems, 2015). A similar feature is implemented here and can be used by pressing the command or control key while dragging on a modulation control. The modulation values for all active steps between the last clicked control and the command/control-clicked control will linearly change between the values of the endpoints.

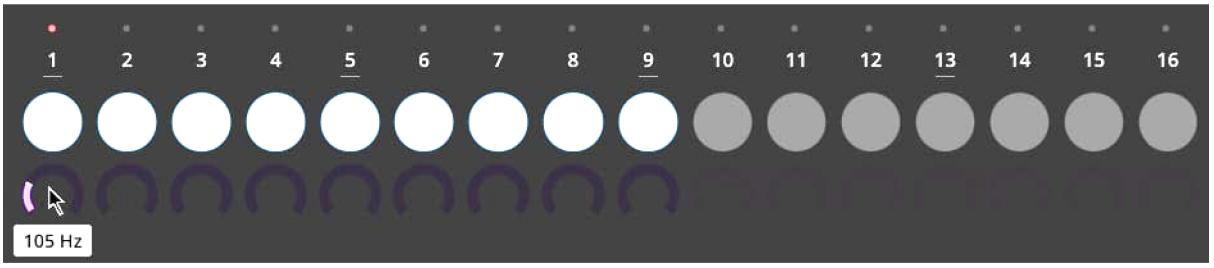

Figure 3.21. To create a 'slope' of modulation values, the value of the start of the slope has to be set first.

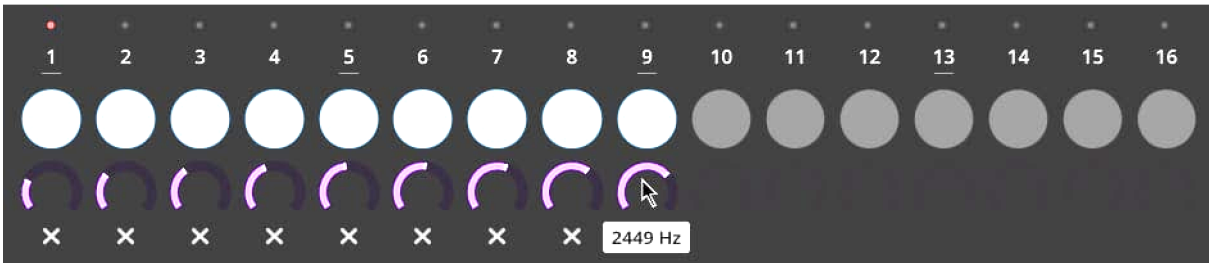

Figure 3.22. Then, when setting the end value of the slope with the modifier key pressed, all modulation values between the start and end will change linearly.

Soloing and muting tracks is a typical operation for fine-tuning a specific sound or group of sounds. The shortcuts for soloing and muting the currently selected track are the S and M keys respectively. Other tracks can also be soloed and muted by engaging the shortcuts while hovering the cursor on one of the track tabs. Sometimes it is easier to edit a sound's parameters by triggering the sound on demand instead of playing a soloed sequence. For this purpose, the enter-key will trigger the currently selected track's sound. Like the solo and mute shortcuts, the enter-shortcut also works for other tracks when the cursor is hovering on their tabs. The scrolling mouse gesture will change the value of a parameter when a control is being hovered with the cursor. Otherwise, scrolling will change the main output volume of the currently selected track. The volumes of other tracks can also be changed by, again, hovering the cursor on a tab and scrolling.

Steps can be selected by clicking on the step numbers, and when a selection is active, changes to the modulation values and active state will be repeated for the selected steps only. To help make regular selections, selecting a step by holding down the command or control modifiers will extend the current selection to the entire sequence. This function is inspired by a similar one in the Cirklon sequencer called 'ganging' steps (Sequentix Music Systems, 2015).

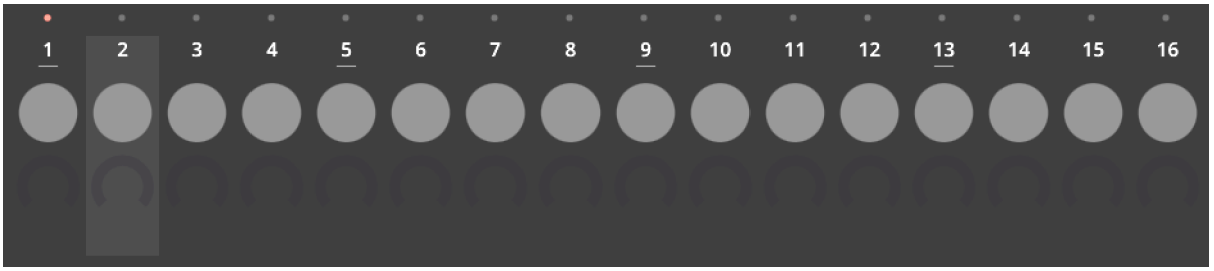

Figure 3.23. To select all even steps, first step two is selected. Selected steps are represented with a lighter background.

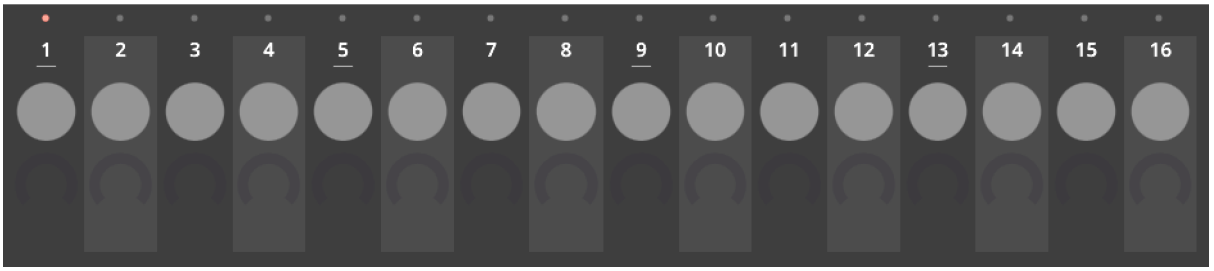

Figure 3.24. Next, step three is selected with the command/control modifier, extending the selection pattern before step three across the entire sequence.

3.8. Randomisation and macros

To aid users in arriving at unexpected results, two randomisation buttons are implemented for randomising both the active steps and the modulation values of the selected parameter. When steps are selected, the modulation randomiser will randomise all locked parameters on only the selected steps instead. Scaling and offsetting is available for quickly adjusting all modulation values at once. By scaling, all modulation values of a parameter will either move towards or further away from the current value of the parameter relative to how far off they are at the beginning of the action. In contrast, offsetting all modulation values of the selected parameter will move the modulation values in the same direction by the same amount. These macro controls are handy, for example, when all modulation values have been randomised and wanting to further tame or fine-tune the values. A dedicated macro is also available for adjusting parameter values: holding down the command or control key while changing a parameter will change the same parameter across all tracks by the same amount. This kind of a ‘control-all’ feature is inspired by a similar function in the Digitakt (Elektron, 2020b). For more usability-focused macros, a button for clearing all steps from a sequence is available, and another one allows clearing all step-modulation for the selected parameter. The macros presented in this paragraph aim to both help create a fluent workflow by making common operations faster, and to increase serendipity.

3.9. The sample and pattern library

The sample and pattern library is a simple, dedicated file system for easier access and organisation of samples and patterns. It runs in a browser database located entirely on the user's computer and is the primary mechanism for loading samples onto tracks and switching patterns. Samples and patterns can be imported from the user's computer, and patterns can also be downloaded in a unique file format. In addition, items can be deleted, renamed, and organised into virtual folders. The application supports shipping with a set of default samples and patterns which are organised into folders according to their creator. Currently, the default library only consists of a small set of very rudimentary samples created by myself. However, in the future, it would be possible to include sample packs by different artists crediting each artist appropriately. When in a default folder, the author's information, including their name, picture, description, and a website link, can be displayed at the top of the library to add a personal touch to the default sample packs.

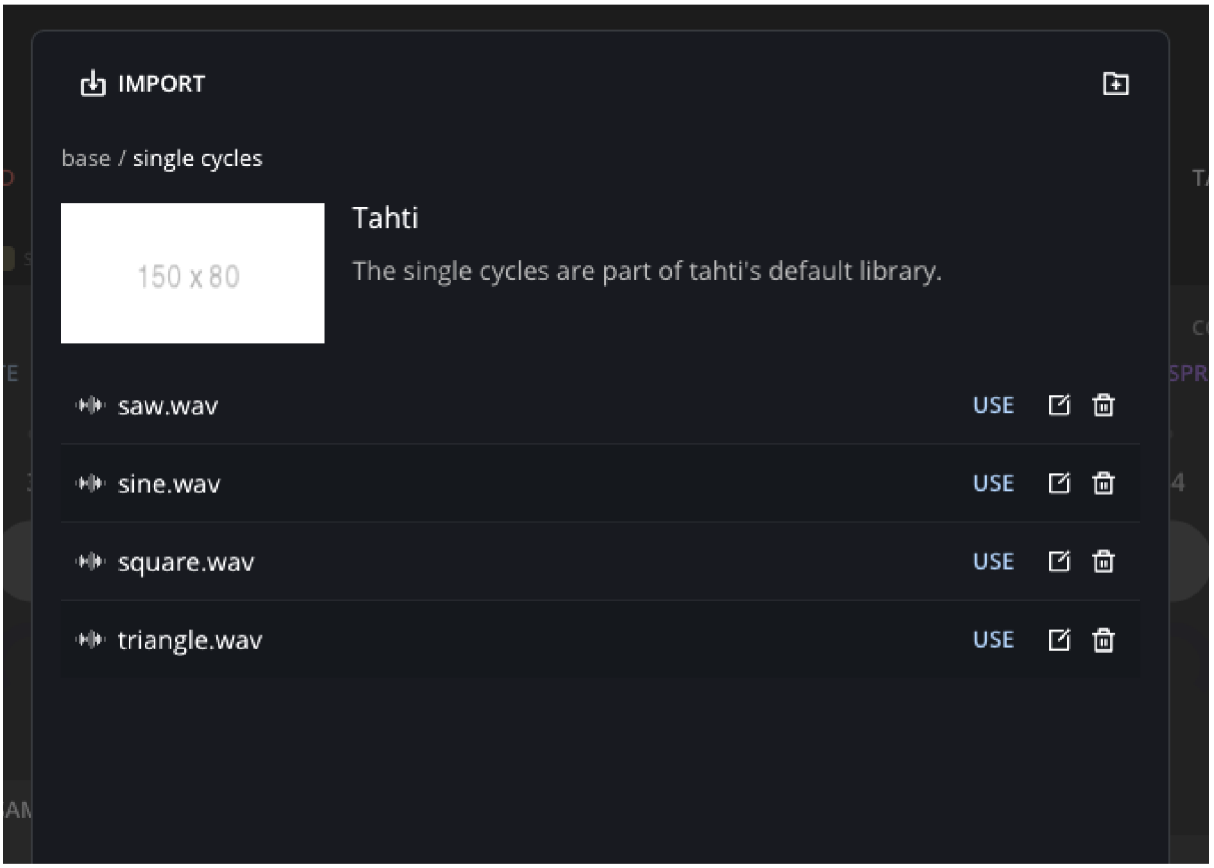

Figure 3.25. Screenshot of the sample library. Special author folders can have custom descriptions and pictures highlighting the creator of the samples.

The unique pattern format embeds all used samples to make sharing patterns as easy as possible. However, including the samples in the file called for some optimisations for keeping the file size at bay. At its core, the format is simply a JSON (JavaScript Object Notation) object compressed with a JavaScript port of the gzip compression algorithm (*pako*, 2021). The algorithm was chosen because it is widely used on the web, and available implementations were easy to find and use. In theory, a newer compression algorithm called Brotli offers even better JSON compression (Shurtz, 2019), but after trying a couple of client-side JavaScript implementations and not getting them to work, Brotli was abandoned for the time being. Potential gains in JSON compression might anyway be negligible compared to the size of the embedded audio samples. The audio data is converted to the FLAC (Free Lossless Audio Codec) format before embedding it in the file to further decrease its size. This was achieved with the open-source libflac.js library, which uses a dedicated WebAssembly module for the conversion (*libflac.js*, 2021).

4. Future work

In this chapter, the final results are evaluated by presenting future development opportunities. Each sub-chapter focuses on a particular aspect of the application, exploring potential improvements. The discussion does probably not cover every possibility for future work exhaustively but is instead a list of the most apparent shortcomings from the author's perspective.

4.1. Social interactivity

As the application is accessible directly via the web, it would be natural to build features that enable sharing creations and collaboration between users. One of the first things could be a feature that allows creating a unique link to a pattern that could be used to easily share patterns. Such a feature would also lead to the question of whether patterns should be versioned so that changes by different collaborators could be tracked and evaluated. Even an entire community with user profiles could be developed by sharing creations and reacting to them using common social media paradigms like commenting and liking. As the application is currently static with no server-side dynamic content, it can be hosted for free on specific services (*Vercel*, no date), but building even the most simple link-sharing functionality would already require some sort of custom server which will introduce monetary costs. To make social interactivity features feasible, some kind of monetisation plan needs to be developed first to cover the required server infrastructures' costs. A more cost-effective and straightforward method to develop communities around the application would be to leverage existing social media platforms, like Reddit or Discord. For example, a dedicated subreddit could be created on Reddit, and a dedicated chat-server in Discord, where users could share pattern files.

4.2. DSP algorithms

The digital signal processing algorithms developed during this thesis are, in general, still very rudimentary and unoptimised. Most room for improvement on the sound lies within the filter, distortion, and reverb algorithms. Other DSP algorithms are, of course, not perfect either, and many of them might also benefit from further development. However, the ones discussed next are likely the most inadequate.

All filter models are based on formulas that are generally more clean-sounding than characterful and were selected for this application due to their availability and ease of implementation. However, by designing optimal filters specifically for each application, considerable sonic and performance improvements are likely achievable. The filter is

arguably one of the most critical parts affecting each track's sound, so other alternatives should be explored.

The distortion algorithms were also selected because they were readily available and would benefit from further development and testing. Because distortion introduces a lot of high-frequency content, it is vital to oversample the distortion signal path to avoid aliasing. (Pirkle, 2014) This is not yet implemented in part due to performance considerations but might be a valuable improvement for the future.

Good-sounding reverbs are notoriously difficult to design, as is evident in testimonials by ValhallaDSP developer Sean Costello, who has been designing reverb algorithms for the past 12 years: "At least 90% of my reverb algorithms have sucked big time. However, I'm obsessed with the stuff, so I've created several hundred reverb algorithms in the past decade+. The algorithms I have released are some of the ones that don't suck." (Costello, 2011). Due to this difficulty, most good-sounding algorithms are commercialised and few are readily available to be implemented (Costello, 2015). Although passable for some applications, the current reverb implementation is probably the weakest single DSP algorithm and would benefit from further development or replacement by a completely different internal structure. Reverb is an essential part of any groovebox, so this is important for future work.

A master compressor is completely missing in the application, although it is found on many hardware grooveboxes, including the Digitakt, Analog Rytm, Octatrack and SP-16. A master compressor, in particular, is useful for controlling the dynamics of the final output, glueing all of the tracks together, and would be an essential addition to support creating more polished-sounding compositions. However, developing a suitable compressor is no easy task and was therefore left out of this thesis's scope.

4.3. Support for extended compositions

Pattern chaining and even more sophisticated ‘song mode’ features are found on several hardware grooveboxes. They are considered crucial functionality by many (‘Search results for “song mode” relating to Digitakt’, 2020), mainly because it allows to create compositions with multiple different sections. While patterns can currently be hot-swapped even when playback is engaged, it is only an accidental feature instead of intentional functionality. Switching patterns might currently cause glitches as sample buffers are suddenly replaced, for example. There is also no support for quantised pattern changes or specifying pattern chains to create longer and more varied compositions. Support for either pattern chaining or a more fully-featured song mode is clearly an important improvement but would require considerable development as it requires deeper architectural changes to the system.

4.4. Porting the application to other platforms

Having a version of Tahti on different platforms opens up exciting possibilities. For example, compositions could be created in a browser or on a mobile version and then brought into a plug-in version running in a DAW for finishing touches. To keep the scope reasonable, the application has so far been only designed to be used on desktop computers with a keyboard and a mouse as input devices. Nevertheless, as the audio engine is wholly implemented in SOUL, it would be straightforward to compile it to different targets like native code for mobile devices or audio plug-ins. However, the audio engine is useless without a well-functioning user-interface, as discussed in chapter 4.1. Here the entire interface is implemented using web technologies, including HTML, CSS and JavaScript, which require a browser as their environment. While it is possible to embed browsers in mobile applications and even audio plug-ins, the performance and binary size usually become bottlenecks. Bringing the application to environments outside of the browser would mean porting the user interface using other platform-specific technologies, which is a remarkable undertaking but nevertheless possible.

The current user interface features could be more or less directly ported to a desktop plug-in version, but for a mobile version, the entire interface would have to be redesigned from a touch device’s perspective. As most internet users are nowadays on mobile devices, a mobile version would make much sense as a goal for future development (*Desktop vs Mobile vs Tablet Market Share Worldwide*, 2020). As stated by Roberts et al. (2013):, “Ideally, a web-based musical interface should be able to run on desktop machines, laptops, and mobile devices regardless of the browser or operating system it is presented in.”. This is indeed an ideal situation but hard to accomplish in

practice. As many features are currently best accessed via keyboard shortcuts, a mobile version would have to develop several alternative paradigms to keep the usability at the same level. Due to the limited screen estate, the layout would also have to be very different for touch areas to have a sufficient size. A notable technical blocker preventing mobile-accessibility is the limited support of required WebAudio technologies in mobile browsers, notably on iOS devices (MDN, 2021). Also, the performance of audio-focused WebAssembly modules on mobile devices would have to be benchmarked; it is not certain that they provide equal performance levels as on desktop devices.

4.5. Communication with external systems

Modern browsers (except for Safari) support MIDI input and output (W3C, 2015), which could be used to synchronise the tempo and transport controls to external MIDI-enabled systems like hardware gear. MIDI could also be used to play the sample tracks and control the parameters externally either for live improvisation or for recording data into the sequencer. For synchronising playback, also support for Ableton Link could be considered (Ableton, 2020b). However, as browser audio is in its early stages, the latency for such extensions might still be unbearably high and would have to be studied more closely (Adenot, 2016).

4.6. Graphical design and user-experience

Although I care a lot about the aesthetics and usability of artefacts, I have no experience or qualifications as a graphical or user-experience designer. While the application’s graphical design and usability are arguably passable, these areas would benefit from further refinement. In addition to improving the graphical design, the user-interface should be tested by conducting proper usability tests with potential end-users to reveal problematic aspects that might have turned into blindspots during extended development. Letting a colleague of mine test an earlier version of the application for just a few minutes already revealed a problem with the slider-like design of the controls of that version, which prompted me to change the controls to their current knob-like appearance. This experience is, of course, anecdotal, but I think it is reasonable to expect more significant gains with proper systematic user testing.

5. Conclusions

The thesis described the process of designing and implementing a browser-based groovebox. The aim was to implement an application that could be compared to existing hardware grooveboxes, which was mostly accomplished by taking a substantial amount of design ideas from a couple of hardware devices. Eight tracks, a healthy set of sound processing options, a flexible sequencer, and various modulation features were implemented. As almost every parameter can be modulated both via the step-sequencer and using LFOs, the creative potential is vast. Even some features usually omitted from hardware grooveboxes, like the sample spread parameter, the frequency shifter, and arbitrary individual sequencer speeds were included in the design as I deemed them interesting based on personal experiences. Luckily, performance did not become too much of a bottleneck for implementing the desired features, although some optimisations were necessary.

The selected technologies, namely SOUL and React, turned out to be a very productive fit for implementing the application. As both are geared towards writing highly modular components, working with the code for an extended period did not become painful. A couple of bigger internal refactorings were necessary along the way because both technologies require adhering to their paradigms pretty strictly. This applied especially to SOUL, as I had practically no experience with the language when starting the project. However, in terms of code clarity, these refactorings were definitely beneficial considering the eventual result and were crucial for future-proofing the application's code. Although the performance would have to be studied more closely and benchmarked against code written with other languages, it seems that SOUL was able to provide rather good performance for the audio engine.

Although already quite usable, the application's development is still in its early stages. When compared to hardware grooveboxes, the most apparent missing feature is chaining patterns, which indeed limits the creative potential of the application considerably. Another aim and design principle was to strive for intuitiveness for first-time users while providing enough depth for long-time users. While there is certainly a lot of depth and creative potential, the onboarding experience of unknowledgeable users could still be improved. In addition, the review of existing hardware grooveboxes in chapter 2.2 was quite brief, and more grooveboxes could be studied for further inspiration. However, I believe the selected devices provided enough of a foundation considering the scope of the thesis. Extending the application to other platforms, like mobile phones or plug-ins inside of a DAW, is one of the most exciting possibilities for future development. Being able to easily start a musical idea on a mobile phone or any computer, and then bringing it into a fully-fledged music

application for finishing it is a compelling thought. However, as discussed more thoroughly in the previous chapter, there are also more pressing requirements before even the first version of the application could be made public, like developing improved DSP algorithms, building features for sharing patterns, and finding and fixing bugs and other illogical behaviours. The designed application is not yet stable enough for use by an extended audience, but it is a solid foundation for future work. Despite some of its shortcomings, the application is already considerably more advanced than all other groovebox-like tools that were examined in chapter 2.1. Personally, I am very content with the result, and I often find myself immersed in the application, exploring its sonic possibilities. Interesting results occur almost every time I delve into Tahti.

References

Ableton (2020a) *Ableton Reference Manual Version 10*. Available at: https://www.ableton.com/en/manual/live-instrument-reference/#the-pitch-envelope (Accessed: 18 January 2021).

Ableton (2020b) 'Link Documentation'. Available at: https://ableton.github.io/link/ (Accessed: 17 January 2021).

Adenot, P. (2016) 'Web Audio API performance and debugging notes'. Available at: https://padenot.github.io/web-audio-perf/#latency (Accessed: 17 January 2021).

Aisher, B. (2015) 'Character & Crunch: Replicating The Sound Of Vintage Samplers', *Attack Magazine*, 14 August. Available at: https://www.attackmagazine.com/technique/character-crunch-replicating-the-sound-of-vintage-samplers/ (Accessed: 15 February 2021).

Akai (2020) 'Akai MPC User Guide'. Available at: https://www.akaipro.com/mpc-one (Accessed: 5 January 2021).

Anderson, J. and Costello, S. (2009) 'ADAPTING ARTIFICIAL REVERBERATION ARCHITECTURES FOR B-FORMAT SIGNAL PROCESSING', in. *Ambisonics Symposium*, Graz, Austria.

Announcing 'Genius Home Studio' Music Creation & Collaboration Program Powered By HP (2020) *Genius*. Available at: https://genius.com/a/announcing-genius-home-studio-music-creation-collaboration-program-powered-by-hp (Accessed: 22 January 2021).

Boden, M. A. (1990) *The Creative Mind: Myths and Mechanisms*. Psychology Press.

Clark, J. J. (2003) *Advanced Programming Techniques for Modular Synthesizers*. Available at: https://www.cim.mcgill.ca/~clark/nordmodularbook/nm_book_toc.html (Accessed: 29 January 2020).

Cook, P. (2001) 'Principles for designing computer music controllers', in *Proceedings of the 2001 conference on New interfaces for musical expression*. SGP: National University of Singapore (NIME '01), pp. 1–4.

Costello, S. (2011) 'Sean of Valhalla DSP Interview'. Available at: https://goodrecordingstuff.blogspot.com/2011/04/sean-of-valhalla-dsp-interview.html (Accessed: 5 January 2021).

Costello, S. (2015) '1000 Years of Reverbs'. Available at: https://valhalladsp.com/2015/06/19/slides-from-my-aes-reverb-presentation/.

Desktop vs Mobile vs Tablet Market Share Worldwide (2020) *StatCounter Global Stats*. Available at: https://gs.statcounter.com/platform-market-share/desktop-mobile-tablet (Accessed: 17 January 2021).

drumbit - online drum machine (2017). Available at: https://drumbit.app (Accessed: 5 January 2021).

Elektron (2020a) 'Analog Rytm User Manual'. Available at: https://www.elektron.se/support/?connection=analog-rytm#resources (Accessed: 5 January 2021).

Elektron (2020b) 'Digitakt User Manual'. Available at: https://www.elektron.se/support/?connection=digitakt#resources (Accessed: 5

January 2021).
Elektron (2020c) ‘Octatrack MKII User Manual’. Available at: https://www.elektron.se/support/?connection=octatrack-mkii#resources (Accessed: 5 January 2021).
Elektron (no date) *Who We Are*. Available at: https://www.elektron.se/about-elektron/ (Accessed: 25 January 2021).
Future Music (1996) ‘Roland MC-303’, May.
Genius Home Studio (2020). Available at: https://homestudio.genius.com/ (Accessed: 5 January 2021).
Korg (2014) ‘electribe Owner’s Manual’. Available at: https://www.korg.com/us/support/download/manual/0/368/2368/ (Accessed: 5 January 2021).
libflac.js (2021). DFKI MultiModal Interaction Group. Available at: https://github.com/mmig/libflac.js (Accessed: 19 February 2021).
Magnusson, T. (2010) ‘Designing Constraints: Composing and Performing with Digital Musical Systems’, *Computer Music Journal*, 34(4), pp. 62–73. doi: 10.1162/COMJ_a_00026.
MDN (2021) *AudioWorklet - Web APIs*, *MDN*. Available at: https://developer.mozilla.org/en-US/docs/Web/API/AudioWorklet#browser_compatibility (Accessed: 9 March 2021).
pako (2021). Nodeca. Available at: https://github.com/nodeca/pako (Accessed: 19 February 2021).
Pioneer (2019) ‘TORAIZ SP-16 Manual’. Available at: https://www.pioneerdj.com/en/product/production/toraiz-sp-16/black/support/.
Pirkle, W. (2013) ‘Virtual Analog (VA) Filter Implementation and Comparisons Will Pirkle’, p. 24.
Pirkle, W. (2014) *Designing Software Synthesizer Plug-Ins in C*. 1st edition. Focal Press.
Pirkle, W. C. (2019) *Designing Audio Effect Plugins in C++: For AAX, AU, and VST3 with DSP Theory*. 2nd edition. New York, NY: Routledge.
Polyend (2020) ‘Polyend Tracker Manual’. Available at: https://polyend.com/tracker/#downloads (Accessed: 5 January 2021).
Redmon, N. (2012) ‘A one-pole filter’, *EarLevel Engineering*. Available at: https://www.earlevel.com/main/2012/12/15/a-one-pole-filter/ (Accessed: 23 February 2021).
Roberts, C., Wakefield, G. and Wright, M. (2013) ‘The web browser as synthesizer and interface.’, in. *International Conference on New Interfaces for Musical Expression*, Graduate School of Culture Technology, KAIST, Daejeon, Republic of Korea.
Rogers, C. and Moore, K. (2012) *WebAudio MIDI-fied Drum Machine*. Available at: https://webaudiodemos.appspot.com/MIDIDrums/index.html (Accessed: 17 January 2021).
Roland (2019) ‘Roland MC-707 Reference Manual’. Available at: https://www.roland.com/global/support/by_product/mc-707/owners_manuals/ (Accessed: 5 January 2021).
Roland (2020) ‘Roland TR-8S Reference Manual’. Available at: https://www.roland.com/global/products/tr-8s/support/ (Accessed: 23 February 2021).
Roland 808303.studio (2020). Available at: https://808303.studio (Accessed: 5 January

2021).
ROLI (2020a) *SOUL*. soul-lang. Available at: https://github.com/soul-lang/SOUL (Accessed: 30 October 2020).
ROLI (2020b) 'The soul command-line tool'. Available at: https://github.com/soul-lang/SOUL (Accessed: 5 February 2021).
ROLI (2020c) 'The SOUL Patch Format'. Available at: https://github.com/soul-lang/SOUL (Accessed: 5 February 2021).
ROLI (2021) *SOUL filter library source code*. ROLI. Available at: https://github.com/soul-lang/SOUL (Accessed: 17 January 2021).
Ross, B. (1987) 'Happy Accident', *The Joy of Painting*.
'Search results for "song mode" relating to Digitakt' (2020). Available at: https://www.elektronauts.com/search?q=song%20mode%20topic%3A39863 (Accessed: 17 January 2021).
Sequentix Music Systems (2015) 'Cirklon User Manual'. Available at: https://www.sequentix.com/cirklon-downloads (Accessed: 18 January 2021).
Shurtz, R. (2019) 'JSON Compression: Alternative Binary Formats and Compression Methods', *Lucidchart*, 6 December. Available at: https://www.lucidchart.com/techblog/2019/12/06/json-compression-alternative-binary-formats-and-compression-methods/ (Accessed: 19 February 2021).
Splice Beat Maker (no date). Available at: https://splice.com/sounds/beatmaker (Accessed: 5 January 2021).
Splice sample search (2021) *Splice*. Available at: https://splice.com/sounds (Accessed: 22 January 2021).
Vercel (no date). Vercel Inc. Available at: https://vercel.com/ (Accessed: 17 January 2021).
W3C (2015) 'Web MIDI API'. Available at: https://www.w3.org/TR/webmidi/ (Accessed: 17 January 2021).
W3C (2021) 'AudioWorkletNode - Web Audio API'. Available at: https://webaudio.github.io/web-audio-api/#audioworkletnode (Accessed: 17 January 2021).
WebAssembly Community Group (2017) 'WebAssembly 1.1 Introduction'. Available at: https://webassembly.github.io/spec/core/intro/introduction.html#design-goals (Accessed: 5 February 2021).
Wessel, D. and Wright, M. (2002) 'Problems and Prospects for Intimate Musical Control of Computers', *Computer Music Journal*, 26(3), pp. 11–22. doi: 10.1162/014892602320582945.
Zafarris, J. (2017) 'The Etymology of "Serendipity"', *Useless Etymology*, 2 December. Available at: https://uselessetymology.com/2017/12/02/the-etymology-of-serendipity/ (Accessed: 30 January 2021).
Zavalishin, V. (2015) *The Art of VA Filter Design*.