U.PORTO

**MESTRADO**
MULTIMÉDIA - ESPECIALIZAÇÃO EM SOUND DESIGN E MÚSICA INTERACTIVA

# LIVE INTERFACE FOR GENERATIVE RHYTHM SEQUENCING

Nuno Diogo Vaz Loureiro de Oliveira

**M**

**2018**

FACULDADES PARTICIPANTES:

**FACULDADE DE ENGENHARIA**
**FACULDADE DE BELAS ARTES**
**FACULDADE DE CIÊNCIAS**
**FACULDADE DE ECONOMIA**
**FACULDADE DE LETRAS**

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

# Live Interface for Generative Rhythm Sequencing

**Nuno Diogo Vaz Loureiro de Oliveira**


Mestrado em Multimédia da Universidade do Porto

Orientador: Prof. Doutor Rui Penha

Coorientador: Prof. Doutor Gilberto Bernardes

July 19, 2018

# Live Interface for Generative Rhythm Sequencing

**Nuno Diogo Vaz Loureiro de Oliveira**

Mestrado em Multimédia da Universidade do Porto

Aprovado em provas públicas pelo Júri:

Presidente: Prof. Doutor Alexandre Valle de Carvalho
Vogal Externo: Prof. Doutor José Alberto Gomes
Orientador: Prof. Doutor Rui Penha
July 19, 2018

# Abstract

Traditional rhythmic sequencing techniques are often not ideal to program complex evolving patterns as they offer only linear control to the player. Techniques available to generate variations of a programmed rhythm usually rely on simple randomness or complex programming actions that do not favor the sequencer as a live playing instrument from the musician's perspective. With this in mind, our idea was to create an interactive system able to generate rhythmically informed variations of a pattern previously entered by the user in a direct and familiar real time performative manner, by means of meaningful generative descriptors providing nuanced control over the complex rhythmic sequencing. To this end, Rhythmicator, a Max/MSP application that automatically generates rhythms in real time in a given meter was used to tackle the generative process around the sequence written by the user. The development of the system is based on the Pure Data programming environment, having some parts of Rhythmicator's Max/MSP code been translated and used for this project. A MIDI controller is used to interact with the system's Pure Data patch and MIDI triggers are sent to any MIDI-able device intended.

**Keywords**: generative rhythm, performative sequencing, real time, rhythmicator, pure data, stochastic model, barlow, metric indispensability, physical control

# Resumo

As técnicas de sequenciação ritmica tradicionais nem sempre proporcionam o ambiente ideal para programar padrões complexos e em constante mutação pela natureza linear do seu controlo. As técnicas disponiveis para a geração de variações de um ritmo programado são vulgarmente sustentadas em simples processos de aleatoriedade ou requerem um processo complexo de programação ritmica que não joga a favor do sequenciador enquanto instrumento performativo do ponto de vista do músico. Assim, surge a ideia de criar um sistema interactivo capaz de gerar variações ritmicamente informadas de uma sequência criada pelo utilizador, através de descritores siginificativos que possibilitam um controlo performativo, de forma directa e familiar, em tempo real, da sequenciação ritmica. Para o efeito, foi utilizado o Rhythmicator, uma aplicação feita em Max/MSP que faz geração automática de ritmos em tempo real consoante um determinado compasso, fazendo a ponte entre a sequência escrita pelo utilizador e o processo generativo. O desenvolvimento do sistema foi feito no ambiente de programação Pure Data, tendo algumas partes do código de Max/MSP do Rhythmicator sido traduzidas para o projecto. Um controlador MIDI é utilizado para interagir com o patch de Pure Data do sistema que, por sua vez, envia mensagens MIDI para qualquer instrumento que o aceite.

**Keywords**: ritmo generativo, sequenciação performativa, tempo real, rhythmicator, pure data, modelo estocástico, barlow, indispensabilidade métrica, controlo físico

# Agradecimentos

ao Professor Doutor Rui Penha pela orientação, disponibilidade e acompanhamento ao longo do mestrado.

ao Professor Doutor Gilberto Bernardes pelo interesse e ajuda no arrancar do projecto.
ao Professor Doutor Georgios Sioros pelo desenvolvimento do Rhythmicator.
ao Diogo Cocharro pela explicação essencial das entranhas do Rhythmicator.
ao Ramires pela ajuda e habilidade em C++.
ao Valter Abreu pelas multiplas ajudas ao longo de todo o processo.
ao Nuno Castro pelos conselhos e pela constante disponibilidade para ajudar.
ao Gui, ao Vasco, aos FUGLY e a todos os meus amigos que contribuiram para este projecto de uma maneira ou de outra.

aos meus Pais.

à Patrícia.

Nuno Loureiro

# Contents

# List of Figures

# Chapter 1

# Introduction

## 1.1 Motivation

Musical sequencers exist under various formats oriented to different needs. It is common for a musician to favor a certain style of sequencing regarding the intended output for a given piece. For rhythmic purposes step sequencers are the most common, be it on a hardware or software environment. Although this approach is very capable of reproducing a sequence the user has in mind, it is limited to the users input and works under a very deterministic set of parameters.

Generative rhythmic pattern generators have been greatly developed and implemented on various forms of musical production, using visual programming languages softwares, such as Max[1], Pure Data[2], or as plug-in extensions in Digital Audio Worlstations (DAWs). Hardware rhythmic sequencers tend to function in a more conservative manner, with the step system as its most common interface. There are, of course, exceptions, especially in Eurorack[3] form, where various techniques, like probability or topographic sequencing, for instance, are used and generative results are produced. This is covered on the next chapter.

## 1.2 Description of the Work

The initial idea for this project came from a personal need for a drum sequencer that would function as a directed player, in the sense that the user writes the core of what is one particular pattern and then, under a set of rules, the sequencer is free to interpret and play variations of it.

The proposed system aims to make the control of generative rhythmic patterns more tangible to the user - starting from a familiar deterministic standpoint and extending it with generative possibilities in a powerful and expressive manner.

[1] www.cycling74.com
[2] www.puredata.info
[3] modular synthesizer format introduced by Dieter Doepfer in 1996

The ultimate goal is to create a hardware sequencing system that answers these needs:

- direct and familiar pattern-based interface
- meaningful generative descriptors
- possibility for nuanced and strong rhythmic variation
- immediate complex rhythmic generation

Available solutions were studied with regard to their generative method and user interface. Georgios Sioros' work on Rhythmicator[4] was used as the basis for the rhythmic generation as it provided a simple interactive experience for the user and a powerful generative engine. Rhythmicator and other solutions are addressed in more depth on the next chapter.

## 1.3 Structure of the Dissertation

This chapter introduces the theme and motivation behind the work that is being presented.

On chapter 2, the state of the Art is presented. To contextualize the development of a system that is intended to break away from the computer, a brief introduction about laptop music and performance since the 90s and how a candid counter movement is appearing by the hands of people who want a closer relation with their music instruments is presented. Literature research was centered around two main aspects: rhythmic generation and rhythmic sequencing. It was important to analyze the related work that had already been done in order for the system to feel tangible and expressive to the player, knowing what sequencing techniques were available and how to accomplish a similar result to the one purposed here.

Next, on chapter 3, a walk-through of the system's process, from the moment the user inputs a rhythmic sequence until its generative variations are played, is presented. Issues about the system's development on Pure Data and its MIDI control are addressed in Appendix 1.

Chapter 4 shows some examples of how the system transforms the user's original sequence.

Finally, in the conclusion - chapter 5 - my personal evaluation of the system is presented, as well as what can be done in the future to further its development.

[4] www.smc.inesctec.pt/technologies/rhythmicator/

# Chapter 2

# State of the Art

## 2.1 Introduction

In this chapter, I am going to briefly address the laptop music movement and how it has started to give way to a counter movement where performers are keen to work with more tangible interfaces, both in the studio and in live situations. Next, I am reviewing key rhythmic concepts, which are fundamental to understand the sequencing and rhythmic generation involved in this work, and address music sequencing techniques, along with some methods for algorithmic generation of rhythms and present some relevant related work, primarily focusing on the Georgios Sioros' Rhythmicator[1].

## 2.2 A step away from laptop performance

Since the 90s, the laptop has gained a major presence in electronic music - the term electronic used in a broad sense and not directly in regard to dance music. As a powerful tool not only to control musical devices but to process audio as well, it has become a more than capable portable and playable studio for a large variety of music artists and performers.

As a musical instrument, it provides the artist a vast and very diverse set of options to produce sound, which are usually not expensive and in some cases even free. Among these options lies the chance for an artist to develop a device specifically tailored to his/her needs. This can be made possible using various programming environments, such as Max, Pure Data, SuperCollider[2], or others, and goes a long way in the artistic development of an own voice by making the tools for the work him/herself. This will to fabricate the tools for the craft is, of course, common to other techniques, be it analog or digital, but the computer offers a virtual limitless quality to what its user can invent.

[1] http://smc.inesctec.pt/technologies/rhythmicator/
[2] https://supercollider.github.io/

Following the ideas of the Futurists and John Cage, in the 90s, musicians and sound artists became interested in exploring the digital error: noise, crackles and glitches. Much like the pursuit of imperfections and 'failure' as an aesthetic in the late 20th century art world [6]. Many times using software against its purpose, artists became interested in the computer as a sound instrument like no other and collectives focused on this kind of exploration - then known as glitch - started forming. At the time, German labels Noton and Rastermusik [3] and Austrian label Mego were strong forces establishing a movement focused not exactly on a musical style or genre but more related to the digital emphasis on composing and performing with computers.

The large amount of sound capabilities was not the only characteristic seen as an advantage. Portability was, and still is, a huge deal to traveling artists. With a laptop, a single artist could carry an entire studio to a performance without the logistic issues that it previously meant. This made a big case for sound as an interest by itself, being that in a live laptop performance there is no apparent causality to what the audience is listening. Even if the audience is able to understand that what is being played comes from the computer, it is not possible to understand the origin of each sound.

[8] argues that this separation between sound and causality in laptop music is not derived from a will to dismiss or disregard the sound source, contrary to the acousmatic music idea of focusing only on the sound, without its agent and significance [1]. It is instead natural to the laptop as a sound object which simply does not make it possible for visual interpretation to be comprehensible. In this sense, the laptop can be seen as an instrument to deliver a raw and minimalistic performance of sound by itself.

Live music performance has always been associated with the human body, be it by the physical action required in traditional instruments, communication between musicians and even the way certain instruments effectively make their performers "dance" to play them. This close relation goes against what is seen in a laptop performance. Even before laptops, when tape was the medium used for electro-acoustic and other kinds of music exploration, this was already a problem for composers. It often led to exaggerated performances to try to convey exciting connections between performers and the sound production [2].

This brings to question the performative quality of the laptop as an instrument. With electronic music, traditional instruments often give place to music controllers which provide a degree of interaction between the artist and the computer, that would otherwise be performing the sound generation outside of the realm of what the audience perceives.

Issues surrounding the link between sound and its origin are different in the studio space and in a live performance situation. The studio is a place of experimentation for the artist, where creation is not necessarily limited by performance, as such this lack of comprehensible feedback for the audience is not a problem for artists. Problems with "in-the-box" composing tend to be related to the lack of an adapted physical interface an artist might not have and to the will to write in a more playful manner, moving and touching the physical buttons of dedicated sound objects. The

[3] Noton and Rastermusik were united in 1997 as Raster-Noton, and now, since 2017, divided again, as Noton and Raster-Media

flourishing of the interest in new interfaces for musical expression and live interfaces expressed in communities like the NIME[4] [12] or the ICLI conference, recently organized in Porto[5], provides a testimony to the willingness of going beyond the computer screen.

Despite not being quite the solution in terms of visual comprehension for the audience, the new found interest for modular synthesizers, namely the Eurorack format created in 1996 by Dieter Doepfer, that picked up in the early 2000s, and the growing appearance of grooveboxes[6], drum machines, samplers, pocket synthesizers, seem to make a case for artists who want to express themselves outside of the computer. Even if the audience does not know exactly what is going on, by the movement of the performer turning knobs, sliders, among other things, and an overall active posture not associated with a device that is used for so many things unrelated to music, these dedicated electronic music instruments may transmit a closer relation between performer and instrument and contribute to the understanding of the causality of sound production.

On the other hand, the 2000s also saw people proposing a completely opposite way of providing the audience with causality in laptop music. [13] makes a case for transparency in laptop music performance, urging performers to show their screen during the performance. This led to appearance of the Algoraves [7], live coding performances where the audience sees the artist's screen.

The dawn of the internet age resulted in strong online DIY and hacking communities with an open-source philosophy that paved the way for the development of unique musical devices based on the creator's needs. By combining the fast style of communication in forums of interface developers and users with vast catalogs of information, this communities facilitated advances in custom interfaces [18]. Adding this to the development of very small computers, like the Raspberry Pi[7] or the BeagleBone[8], and it was possible for computers to be "disguised" as a fully dedicated musical instrument while maintaining the processing capabilities they are known for. Recent examples of this are Critter and Guitari's Organelle[9], which runs its DSP completely on Pure Data patches, and Monome's Norns[10].

## 2.3 Rhythm

The purpose of this project is to control generative rhythmic sequences, therefore terms like pulse, tempo, meter, rhythm and syncopation need to be addressed.

All these concepts derive from how music is perceived in time. When listening to music, rhythm cognition often comes from a sensation of regular pulse at a particular rate, tempo. People's ability to clap in sync with a song is a practical demonstration of this pulse [9]. The way

[4] www.nime.org
[5] www.liveinterfaces.org
[6] a self contained instrument for electronic music production
[7] www.raspberrypi.org/
[8] www.beagleboard.org/
[9] www.critterandguitari.com/pages/organelle
[10] www.monome.org/norns/

pulses are accentuated forms a hierarchical grouping of sonic events, where some pulses are perceived as stronger than others, which convey an idea of metrical pattern to the listener.

The idea of a meter in music theory is represented as a time signature. It divides a regular interval of time by a certain amount of pulses. Because of this, the performer is aware of the particular periodicity of that time signature and the different values of each beat in a bar, the first being the most important for the listener to feel the time signature. As an example, in a 4/4 time signature each bar would have 4 pulses while a 3/4 time signature would have 3.

As the listener's experience is normally outside of the composition or performance realm of the piece, the time signature expressed in the music sheet is not necessarily conveyed in the same way the meter is perceived by the listener. Different accentuations and other composition techniques may or may not be used to express various changes in what a regular example of a particular time signature is expected to sound like, leading to metrically ambiguous rhythm if so is intended. Nevertheless, meter and rhythm are connected as a cognitive process in the listener's mind, as meter is identified by regularities on a periodic rhythm [15].

Syncopation is a brief discrepancy of the meter. It can be caused by different accentuation or articulation of pulses as well as other melodic or harmonic changes [10]. For the listener to feel a rhythm as a syncopating rhythm, it needs to contrast with a metric regularity present in the rest of the music. This contrast may be of different nature. It can, of course, be due to a purely rhythmic reason, by a momentary change of the primary character of the meter, for instance, but it can also be caused by changing the stress on strong or weaker pulses. Sioros proposes to see syncopation as a transformation of a non-syncopated rhythm pattern by shifting particular events and accents from their original position to a neighbor one.

## 2.4 Music Sequencing

A music sequencer is a device that stores a pattern entered by the user and performs it back when the user intends. There are various types of music sequencers as different issues need to be addressed to sequence a melodic line or a rhythmic pattern for example. As the focus of this project is rhythm, this analysis is centered on rhythm sequencers. Rhythm sequencers are naturally associated with drum machines and are often called drum sequencers.

The most common types of sequencers are loop sequencers, which simply record and repeat what the user played, and step sequencers, where the user enters a pattern step by step, each representing a pulse of the meter, and the machine will read and play it.

Step sequencers come in different kinds. Some synthesizers, like the Roland SH-101, include sequencers that let the user enter a melodic sequence by playing a note or a rest (silence). Each note/rest represents a step so it is not needed for the user to play the line as it will not record rhythm. When the sequence is done, the user can press play and the sequencer will loop through the steps. Rhythmic step-sequencers work in a different way. Roland's 606/808/909 drum machines are examples of this kind of sequencing. The user chooses what drum part to sequence and then

selects which steps are going to be triggered. On play, the sequencer goes through all the steps, checks if they are selected and triggers accordingly.

Both loop and step sequencers are deterministic approaches by nature. Nevertheless some randomizing functions have been introduced in step sequencers. Some generate small variations of the pattern entered by the user while others provide the user a completely new random pattern. This is present in sequencers like the Arturia Beatstep Pro[11] and Polyend Seq[12] respectively.

Step-sequencers are not limited to a linear format. Make Noise René[13], for example, presents a cartesian step-sequencer. As the steps are organized in a 2 dimensional space, it makes possible for different paths in the sequence. By modulating this path function, it is possible to generate rhythmic variation.

### 2.4.1 Rhythmic Generation Models and Generative Sequencers

Different models for rhythmic generation in real time are available. Some standard procedures for computer generation of music, such as genetic algorithms, and other evolutionary methods, stochastic and non-stochastic models, are briefly addressed on this chapter.

A genetic algorithm evolves a population of potential solutions to a problem in order to solve it. It works using genetic operations, like crossover and mutation until an acceptable solution is found by means of a fitness function [5]. For its ability to search a vast field of possibilities it has been used as a creative tool in music for the development of music sequences in particular. The complexity of aesthetic judgments regarding the outcome of the fitness function makes the real time operation a major concern. A solution for this issue has been adopted by [4] in kin.genalgorithm, where they do not use a fitness function, instead they encode several musical constraints directly in the genetic algorithm's operators, but it nonetheless depends on a previous analysis of MIDI loops to generate a sequence.


Figure 2.1: MI Grids.

Mutable Instruments Grids[14] presents another solution. It navigates through an extensive bank of drum loops divided by 3 trigger sections, one for kicks, one for snares and another for hi-hats, each one with an individual density control. By meddling with the X and Y coordinates of the drum loop map, each drum loop is constantly being combined with the next, resulting in an ever changing rhythmic sequence if the user so desires. While there is not a mutation process happening like in kin.genalgorithm and the user interaction is very simple, it does not provide any specific control over what kind of rhythm is being played other than more or less hits.

[11] www.arturia.com/beatstep-pro/overview
[12] www.polyend.com/seq-sequencer/
[13] www.makenoisemusic.com/modules/rene
[14] www.mutable-instruments.net/modules/grids/

We distinguish stochastic rhythmic processes as being stationary or weighted [14]. In a stationary rhythm the listener perceives an irregular but predictable Gestalt variation, a technique often used in sound-mass textures using granular synthesis. A weighted rhythmic process implies an imposition of movement in the stochastic texture. This imposition can be introduced by varying the density of events, amplitude, bandwidth, spectral centroid or other perceptible time-varying operation. Stochastic models may be implemented in a very simple manner, like throwing a dice or a coin in order to select a rhythm pattern, as Cage did, or using more complex algorithms.

Tip Top Audio Trigger Riot[15] is a probability sequencer that fits into this complex stochastic generation category. It functions as a 4x4 matrix of probabilistic control over several parameters such as a clock divider, clock multiplier, step injector, time shifter, clock shifter and pulse-width modulator, giving the user a high degree of control over the sequence introduced.

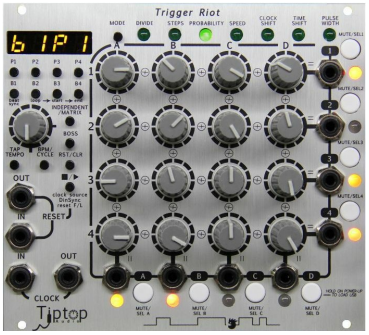


Figure 2.2: Tip Top Audio Trigger Riot.

ADDAC 402[16] is another example of complex generative rhythmic sequencing. It is a 4 voice heuristic rhythm generator with probabilistic and evolutionary modes. It also has an euclidean rhythm mode, among other more common options.

The sweedish manufacturer Elektron[17] is famous for the parameter lock function in their sequencers. This means that the user can set different values for virtually everything in each step on a sequence. Together with triggering probabilities, this function provides a high level of stochastic rhythmic variation while also facilitating changes on the instrument's timbre and effects as well.

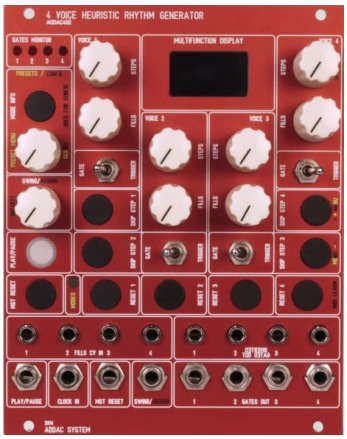


Figure 2.3: ADDAC402.

The options presented above offer a high degree of complexity, nevertheless they come at a cost of a less immediate approach and a steeper learning curve as interaction with the devices is rather complex.

Other non-stochastic approaches exist for rhythm generation like the euclidean algorithm or reflection and toggle rhythms [17]. While this techniques work well for complex rhythmic performance, it is not possible for the user to enter a specific pattern.

[15] www.tiptopaudio.com/trigger-riot/
[16] www.addacsystem.com/product/addac400-series/addac402
[17] www.elektron.se/

#### 2.4.1.1 Metric Indispensability and Rhythmicator

[3] presented an algorithm that outputs the metrical indispensability for a certain meter. It assigns a weight to each pulse according to the importance it has in order for the listener to feel the meter as it is. By providing a metrical hierarchy, this algorithm is capable of flexible results maintaining an interesting balance with musicality.

[16] devised an approach for automatic rhythm generation where Barlow's metrical indispensability rankings are used to generate rhythms specific to the given meter. In this sense, the output of their system is not related to any particular music style, being prone to produce "generic" results within the meter. Real time control of the performance can be achieved by parameters such as density of events, amount of syncopation, amount of variation in generation, the metrical strength of the rhythm being generated and the meter itself. This system was implemented as a Max/MSP abstraction and a Max4Live device called kin.rhythmicator.

Barlow's indispensability algorithm has also been used by [11] to develop a sequencer on Max/MSP that is fully determinate.

Rhythmicator's algorithm is divided in 2 steps. First, it expects the user to enter a musical meter and, according to the specified metric subdivision, subdivides it into the corresponding number of pulses. Each pulse is then assigned a weight value - a ranking measure - depending on its importance in the meter. The second stage takes the weight values and uses them to generate a stochastic rhythm.

Weight calculation is done according to Clarence Barlow's indispensability formula. Depending on the time signature the user introduces, a stratification algorithm distinguishes between simple and compound meters, 3/4 and 6/8, for example. If the user chooses a sixteenth note subdivision both a 3/4 and a 6/8 meter have 12 pulses and so it is important to make the distinction. The algorithm achieves this by decomposing the meter in prime factors and feeding the result to Barlow's algorithm which then delivers the indispensability values for the pulses. The pulse with the highest weight value is considered to be the most indispensable and the lowest to be the less indispensable.

The stochastic performance is generated once the weight values are calculated. Weight values are aligned to the corresponding place in a bar and the probability of an event being triggered is derived from its pulse's weight. The amplitude of triggered events is not related to the probabilities, it is directly proportional to the pulse's weights.

The system produces syncopated rhythms by triggering events in strong metrical positions ahead of time with the probability of the next pulse. Amplitudes are anticipated as well, in order to dynamically accentuate the pulse. To keep the generation musical, syncopation is limited by 2 rules: too many consecutive syncopated pulses cause the system to automatically stop syncopation and, because the syncopated feeling suffers when 2 consecutive syncopated events are triggered, the second pulse will always be mute if the previous one triggered a syncopated event. This second rule is only enforced in case the consecutive anticipated pulses are less than 3 to prevent the cancellation of off-beat syncopation.

The density parameter works by taking action over the probability of each pulse to trigger. This action regards the weight of the pulses in order to maintain the meter's indispensability hierarchy. Density is then directly related to the metric feel. If density is 0, no events are triggered and thus there is not information to infer the meter. If Density is at its maximum, all pulses will trigger and the only way to perceive the meter is the amplitude variation between pulses.

Maximum metrical strength occurs when the events triggered are the most indispensable ones. This depends on the probability of the pulses and the amplitudes of the generated events. As it is, the system triggers important pulses more often and with an higher amplitude, to guarantee metrical stability. Nevertheless, all parameters influence this strength. If pulse triggering probabilities are changed to be similar to each other the meter becomes less evident as a clear hierarchy of pulses ceases to exist.

Although the stochastic nature of the system results in non-repetitive generation of rhythm, it is possible to influence the amount and type of variation by means of the Density and Syncopation parameter. There is also a stable mode that makes the sequencer vary around a random initial pattern and an unstable mode that gets a new random pattern on every cycle.

Rhythmicator's interface consists of a meter selection tool, one slider to control density, another slider to control MIDI note velocity, a switch to select between the stable and unstable modes, a button to re-initialize the rhythmic pattern, a list of probabilities and a circular interface to control the syncopation and variation parameters.

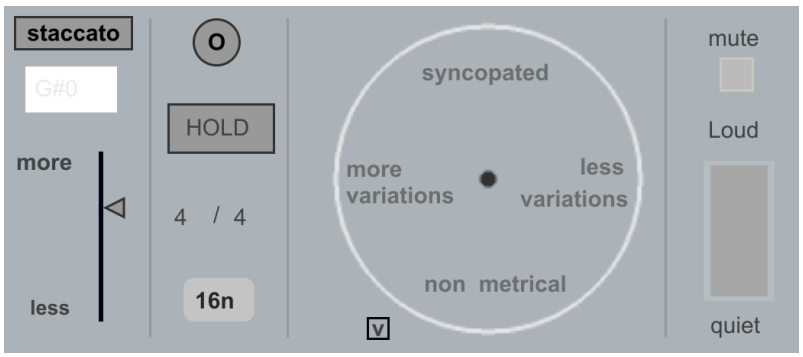


Figure 2.4: Georgios Sioros' Rhythmicator

# Chapter 3

# From the Sequence Input to the Generative Output

## Introduction

We are presenting a generative monophonic trigger sequencer based on the step sequencing technique that can be grouped in multiple instances to form a sequencing system. Sequencing is achieved using the MIDI protocol, but the system can easily be adapted to work in different ways, such as OSC or even CV, with the appropriate hardware tools.

The sequencer has 2 separate stages between the user's input and the rhythmic output. First, it has to identify what the intended basis for the rhythmic generation is, which is done via a familiar step-sequencing technique. On a second part, the rhythmic generation occurs influenced by the sequence the user wrote and a set of parameters and effects available for its real time control.

The sequencing system described here was developed entirely in Pure Data Vanilla with the exception of 2 externals[1], kin_weights and kin_sequencer, built by Georgios Sioros for his work on the Rhythmicator. The reason for this port to Pure Data is to be able to integrate the sequencer in other devices in the future, such as Bela or Raspberry Pi, for example, thus making a complete standalone sequencer.

An Arturia Beatstep Pro was mapped to serve as a controller for this prototype. A brief explanation of the Pure Data patch as well as the settings for the MIDI controller are available on the Appendix A - Implementation.

## 3.1 Sequence Analysis

One of the most important parts of the user interaction in this particular sequencer is the way the user inputs a new sequence. There are many ways of approaching this, as it was discussed before in chapter 2, nevertheless, the one chosen was the step sequencing technique, for its simplicity and overall familiarity within musicians. The user inputs a sequence by simply toggling on or off

[1] external objects developed by third party users of the Pure Data software

each step. Apart from this simple command, the user has the ability to choose how many steps the sequence has.

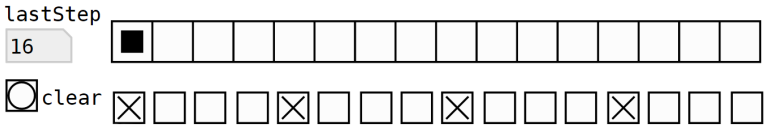


Figure 3.1: Simple 4/4 pattern in sequencer example in Pure Data

### 3.1.1 Time Signature Determination and Indispensability Rankings

For the rhythmic generation to be adequate, the sequencer needs to understand what is the meter of a given sequence. This was thought to happen automatically in order to keep a simpler workflow for the user. The way it works is by analyzing the number of steps present in said sequence. Classifying time signatures on a step sequencer can be ambiguous as each step is a pulse. This means that a sequence with 4 steps, or pulses, corresponds to a 4/4 time signature, if each step is perceived as a quarter note, while a 16 step sequence would mean a 16/4 time signature, which could just be interpreted as 4 4/4 bars or as 1 4/4 bar where each step represents a sixteenth note instead.

Another issue is regarding the tempo driving the sequence. In a musical context, if a sequence with 16 steps, like the one shown before, is felt as a simple 4/4 meter, a sequence with only 4 steps, all of them active, running at a quarter of the tempo would produce the exact same result and metric feel. The only difference being the resolution available for more intricate rhythmic patterns, evident in Figure 3.2.

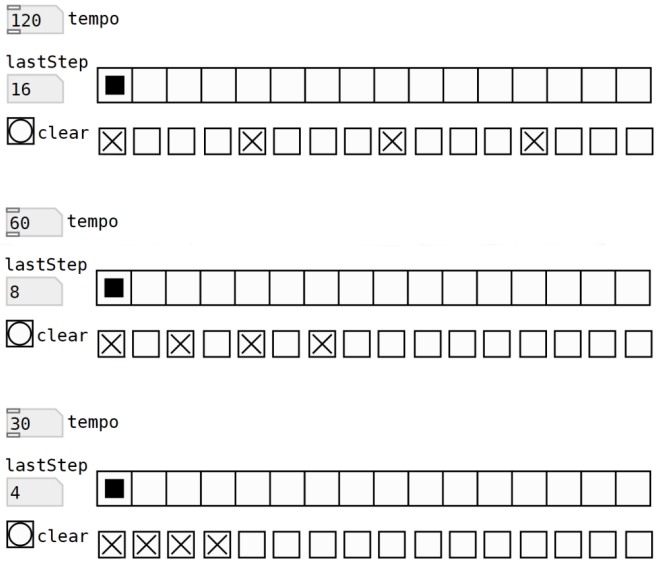


Figure 3.2: The same 4/4 sequence written with different tempo settings

Finally, if one considers that each step represents a quarter note, a sequence with 1 step would have a 1/4 meter, 2 steps 2/4, 3 steps 3/4 and 4 steps 4/4, etc. A problem rises when the sequence has a number of steps that allows not only binary division but also tertiary division, which is the case of a 6 step sequence, for example. In this particular case, the 6 steps could be interpreted as 2 bars of a 3/4 meter or as 1 bar of a 6/8 meter, depending on the sequence, modifying the resolution of the step, from quarter to eighth note.

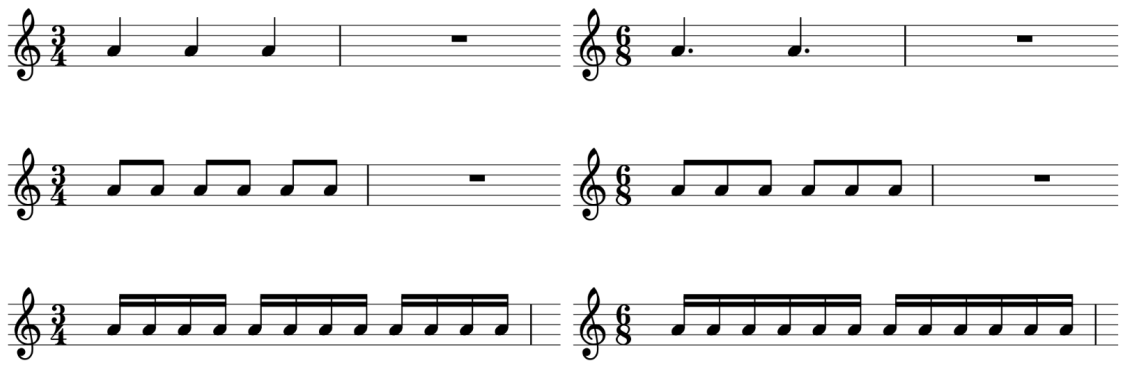

Figure 3.3: Binary and tertiary subdivision of 3/4 and 6/8 Time Signatures

When using a step-sequencer, the user does not need specific information about the musical subdivision each step represents as that is dependent on the relation between the BPM running the sequencer and the whole context of the music being played. As such, time signature is not of particular importance, what is absolutely required is for the sequencer to be able to interpret the metric feel of a given sequence, be it of binary or tertiary subdivision.

There are 3 steps in the sequencer's detection of the suitable meter. First, the user defines the number of steps the sequence should have. Then, a sequence is written. It is important to know that for rhythmic generation purposes, the first pulse of a meter is not necessarily the first step of the sequence. The first pulse of the meter will be determined by the sequencer as it compares the indispensability values of the meter with the pattern written by the user. This happens next as each step of the sequence is multiplied by the according indispensability value. The results of all operations are then summed. Afterwards, the sequence suffers a rotation and goes again, until all possible rotations are performed. The rotation with the highest score will then be chosen as the basis for the rhythmic generation. Sometimes the highest result is shared by more than one rotation, if so, the rotation chosen is the one closer to the original first step of the sequencer.

Figure 3.4 shows this process being applied to a simple 4 step sequencer.

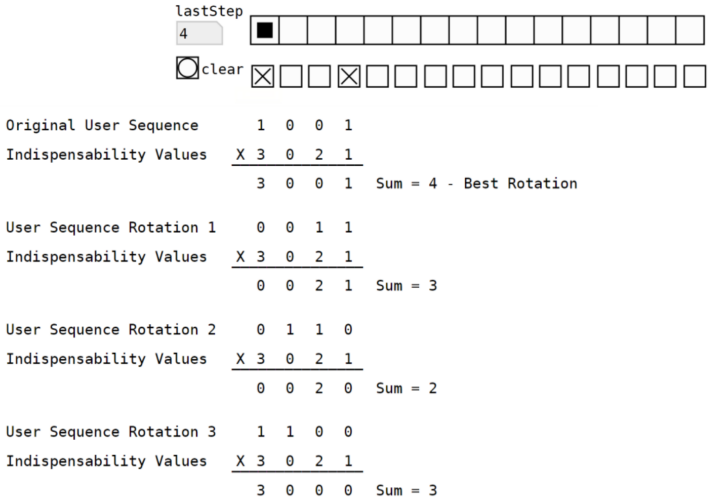


Figure 3.4: Finding the best rotation for a 4 step sequence

In cases where both binary and tertiary division are possible the process runs twice, comparing the sequence to both meter's indispensability values to choose the more appropriate setting. In the example present in Figure 3.5, it is possible to understand that the meter that better fits the sequence is of binary subdivision and that its first pulse coincides with the first step of the sequence.

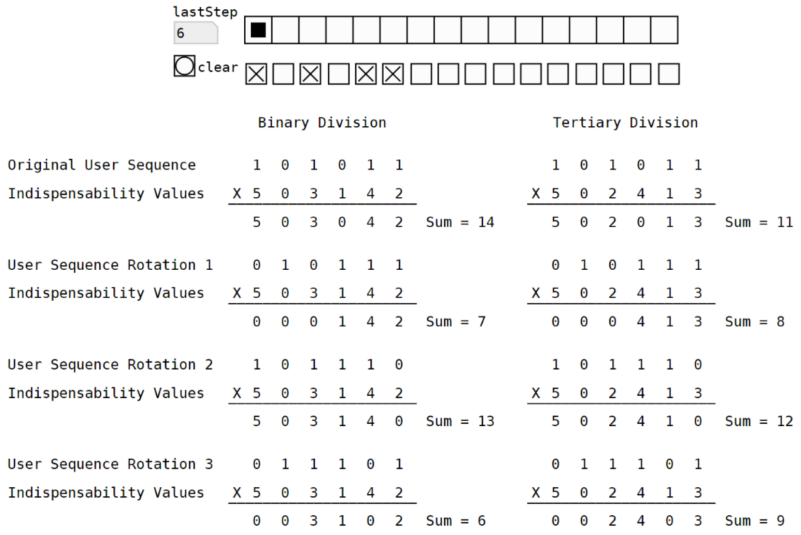


Figure 3.5: Finding the best rotation for a 6 step sequence: binary or tertiary meter

This process takes place every time a step in the sequencer is activated or deactivated as well as when its number of steps is changed. In the end, it provides information about the most appropriate metric feel and rotation of the sequence to the generative engine.

## 3.2 Generative Rhythm Output

Sioros' Rhythmicator functions as an automatic rhythm machine which is not what is intended here. Nevertheless, its capabilities of real time generation and parametric variation are ideal for this approach. For it to work on this system, a transformation from automatic active sequencer to a sequence dependent one must take place.

The Rhythmicator was built around an automatic rhythm sequencer that only needs information about the meter, which it gets by means of the meter's stratification levels. This information is translated into weight values for each pulse, that are then transformed according to a set of control parameters. Weight values are scaled by the Density parameter, increasing or decreasing each pulse's probability to trigger. Therefore, in order to influence the rhythmic generation action must be taken on the weight of selected pulses and on how the parameters command Rhythmicator's sequencer.

### 3.2.1 Rhythmic Parameters

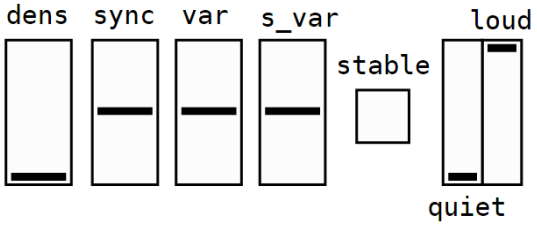


Figure 3.6: Default rhythmic parameters

This system uses the same set of parameters as the Rhythmicator as it serves as the basis for the rhythmic generation. The parameters are more deeply addressed on the Rhythmicator's section of the second chapter - State of the Art. Here, a brief explanation of the user interaction is presented. The user is able to control:

- Density
- Syncopation
- Variation
- Syncopation Variation
- Stability
- Velocity Range

With the exception of the velocity controls, every rhythmic parameter performs its action by setting the kin_sequencer object to different states.

The density parameter is of severe importance to various other parts of the rhythmic generation process, addressed in the following sections of this chapter, but its main function is to control the amount of events that are triggered by cycle. It functions by first influencing the pulses' weight on kin_weights, elevating the probability of an event being triggered according to its rank. Minimum means no event gets triggered and the highest value means maximum density.

Because of the link between metrical feeling and syncopation both parameters are controlled by the same slider - syncopation. From 0% to 50%, syncopation is set to 0% and metric feel goes from 0% to 100%. From 50% to 100%, metric feel is set at 100% and syncopation goes from 0% to 100%.

Highest metrical feeling occurs when all pulses maintain their original weight, the one assigned by the kin_weights object. When the metric feel setting is at 0, all pulses have the same probability of triggering thus ignoring the indispensability ranking.

Sioros' implementation of syncopation works by anticipating events. The probability of an event being triggered before its time is higher according to how high this factor is.

Rhythmic variation is controlled by 2 different sliders. Variation controls the overall variation of the triggered events and syncopation variation is responsible for the amount of variation in syncopating by anticipating different events in each cycle.

Stability control is also available to the user. It is by default activated in order to lock new variations to the written sequence but can be deactivated if the user so intends. This generates completely different variations at each new cycle, although, even if only subtly, because of the pulse weight meddling section of the system, explained next, the user selected pulses still survive the instable variations.

The velocity range controls scale the event's velocity value between a quiet and loud limit, as in minimum and maximum.

### 3.2.2 Pulse Weight Meddling from Sequence Input

Step-sequencers function in binary form, if the step is on it plays, if it is off it does not trigger. Here it is required of the step sequencer to be an influencer rather than a dictator. For that reason, while the steps activated by the user must increase the weight of that particular pulse, the inactive steps' weight should be decreased accordingly to its indispensability ranking - a measure to make sure the automatic generation is a complement of the sequence written by the user and not of the same importance. All of the pulses' weights are, of course, subject to the density of events the user desires.

The density parameter has different behaviors in the 2 systems. In the Rhythmicator, density increases or decreases the preset weights of the pulses, going from 0 to 100, 100 meaning that it is certain for that pulse to be played, maintaining the indispensability hierarchy that characterizes the meter. This new system approaches density in a distinct manner. The density parameter

manipulates selected and unselected steps differently. For better user manipulation, a compromise between the written sequence and the rhythmic generation must be made in such a way that around the middle point of the density parameter the sequencer plays a rhythm very close - number of events triggered - to the one written by the user. Therefore the density parameter had to be configured with that in mind and to give the user better control over lower settings.

Alignment of the user's sequence and meter - indispensability ranking and the standard pulse weight - occurs first. After, the weight of each pulse in regard to the Density setting is calculated, then, the list of weights gets matched with the sequence the user wrote and is influenced by a new scaling of the density value according to whether or not the step is activated. This scaled density (SD) is achieved by first dividing the original range in 2: density (D) set to a value lower than 50 and density set to 50 or over.

If D < 50 then:

$$SD = \frac{0.2 \times \left(\frac{D}{10} - 5\right)^2 + 7}{10}$$

If D $\geq$ 50 then:

$$SD = \frac{3D}{500} + 0.4$$

Next, the scaled density takes part in extracting the final weight of each pulse. There are 2 different functions to calculate this, 1 for selected steps and another for unselected ones. OW refers to the original weight of the pulse and MW to the final meddled weight.

If a step is not selected its meddled weight is calculated by:

$$MW = SD \times OW$$

If a step is selected, the meddled weight is:

$$MW = 20 \times OW \times SD^2$$

In the end, as the weight value has to be between 0 and 1, there is a clipping function, limiting the meddled weights values to that range.

| Indispensability Ranking | | 6 | 0 | 4 | 1 | 5 | 2 | 3 |
|---|---|---|---|---|---|---|---|---|
| Standard Meter Weight | | 1 | 0,571 | 0,857 | 0,643 | 0,929 | 0,714 | 0,786 |
| User Sequence | | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| | Density | | | | | | | |
| Original Weight | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Meddled Weight | | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Original Weight | 12,5 | 0,240 | 0,034 | 0,171 | 0,069 | 0,206 | 0,103 | 0,137 |
| Meddled Weight | | 0,811 | 0,014 | 0,070 | 0,028 | 0,696 | 0,0412 | 0,464 |
| Original Weight | 25 | 0,500 | 0,122 | 0,374 | 0,185 | 0,437 | 0,248 | 0,311 |
| Meddled Weight | | 1 | 0,070 | 0,215 | 0,106 | 1 | 0,142 | 1 |
| Original Weight | 37,5 | 0,740 | 0,270 | 0,583 | 0,348 | 0,662 | 0,427 | 0,505 |
| Meddled Weight | | 1 | 0,179 | 0,389 | 0,232 | 1 | 0,284 | 1 |
| Original Weight | 50 | 1 | 0,496 | 0,832 | 0,580 | 0,916 | 0,664 | 0,748 |
| Meddled Weight | | 1 | 0,347 | 0,582 | 0,406 | 1 | 0,465 | 1 |
| Original Weight | 62,5 | 1 | 0,617 | 0,872 | 0,681 | 0,936 | 0,745 | 0,808 |
| Meddled Weight | | 1 | 0,476 | 0,673 | 0,525 | 1 | 0,575 | 1 |
| Original Weight | 75 | 1 | 0,748 | 0,916 | 0,790 | 0,958 | 0,832 | 0,874 |
| Meddled Weight | | 1 | 0,636 | 0,779 | 9,671 | 1 | 0,707 | 1 |
| Original Weight | 87,5 | 1 | 0,869 | 0,956 | 0,891 | 0,978 | 0,913 | 0,934 |
| Meddled Weight | | 1 | 0,801 | 0,882 | 0,821 | 1 | 0,841 | 1 |
| Original Weight | 100 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Meddled Weight | | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Figure 3.7: Original and Meddled Weights comparison

The indispensability ranking is not only important for the generation of new rhythmic patterns on the sequencer level but is also required for the understanding of the pulses' amplitude hierarchy in the sequence. The amplitude hierarchy accentuates the metric feel and makes possible for events triggered on lighter pulses to be perceived as ghost notes - notes to be heard in the background of the main beat. This can be achieved by MIDI velocity or, as several MIDI drum machines do not have velocity, as a CC message controlling the volume level. There is also an additional level of control where the user can scale the minimum and maximum values of this amplitude parameter.

Regarding the sequencing process, weights are influenced by density on 2 different stages. The first one to set the normal weights - on Rhythmicator's terms - regarding the indicated density and the second to have these weights manipulated by the sequence written by the user. Here, on the amplitude stage, each pulse should maintain the same weight despite the density. The first pulse of the sequence should sound the same whether there is a complex rhythmic pattern playing or if it is the only active pulse, therefore using the density parameter to scale the amplitude weights does not make sense. Nevertheless, it is important to manipulate the weights in order for lighter steps that may have been selected by the user to feel like a strong pulse. This is done by simply multiplying the weight of each selected pulse by 10. Weight values are also limited at 1 in the end.

### 3.2.3 Indispensability Ranking's Rotation

This sequencer automatically adapts the indispensability weights to the sequence the user writes in order to keep the generation in context. In case more than 1 sequencer is running with the same meter at the same time, it is common for the first pulse of the meter to not be the same for every one. Furthermore, one of the sequences is likely to force its first pulse on the listener based on the characteristics of its timbre and the pattern it is playing. A kick drum playing a regular beat will have a tremendous effect on the way the overall meter is perceived, for example. This generates a problem whenever the system is used to generate more than one rhythm.


Figure 3.8: Master, Follower and Rebel Sequencers

The solution found was to have a way to declare a sequencer as a Master to which all the other sequencers would accordingly rotate their pulses' weights. While this corrected rotation produces the expected results in terms of a concrete metric feel, some incorrect rotations can also produce interesting results. As such, a control for the rotation of a sequence was introduced in the sequencer. In case there is a Master sequence, it follows it by default - Follower - and the user is then able to rotate the weights - Rebel. There is a red marker indicating the first pulse of the meter on PD's GUI. If no Master is selected or the Master's meter is different, the sequencer assumes the best rotation of the weights according to the sequence written.

Evidenced in Figure 3.9, the standard weights provide the most suitable match between the introduced sequence and the meter. Nevertheless, it is possible to find other rotations where some of the heaviest pulses match other steps selected by the user, like rotation 4, for example. This results in different accentuations regarding to the overall meter that do not deviate far enough from the norm to be perceived as out of place and thus making for an interesting rhythmic effect.

| Indispensability Ranking | 7 | 0 | 4 | 2 | 6 | 1 | 5 | 3 |
|---|---|---|---|---|---|---|---|---|
| User Sequence | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| | | | | | | | | |
| Standard Weights | 1 | 0,156 | 0,375 | 0,219 | 0,750 | 0,188 | 0,500 | 0,250 |
| Rotation 1 | 0,156 | 0,375 | 0,219 | 0,750 | 0,188 | 0,500 | 0,250 | 1 |
| Rotation 2 | 0,375 | 0,219 | 0,750 | 0,188 | 0,500 | 0,250 | 1 | 0,156 |
| Rotation 3 | 0,219 | 0,750 | 0,188 | 0,500 | 0,250 | 1 | 0,156 | 0,375 |
| Rotation 4 | 0,750 | 0,188 | 0,500 | 0,250 | 1 | 0,156 | 0,375 | 0,219 |
| Rotation 5 | 0,188 | 0,500 | 0,250 | 1 | 0,156 | 0,375 | 0,219 | 0,750 |
| Rotation 6 | 0,500 | 0,250 | 1 | 0,156 | 0,375 | 0,219 | 0,750 | 0,188 |
| Rotation 7 | 0,250 | 1 | 0,156 | 0,375 | 0,219 | 0,750 | 0,188 | 0,500 |

Figure 3.9: Weight rotation on 8 step sequence

### 3.2.4 Generation of Extended Variations through Sequencing Effects

An extra level of rhythmic variation was introduced under the form of a Step Divider. A big limitation of the step-sequencing technique is that events are limited to the same rhythmic figure. This effect makes it possible to transform a step into multiple subdivisions in itself. This means that instead of triggering an event once, the sequencer can trigger a step twice and double the speed, 4 times at 4 times the speed, etc. As an example, if a step is considered to be a quarter note, this effect makes the sequencer trigger 2 eighth notes or 4 sixteenth notes instead of 1 quarter note.

The effect is set up in an singular manner for each step with a probabilistic mind set, in order to get another level of movement in the generation process. After engaging a step, the user can select between 5 different rhythmic subdivisions of the tempo or, if desired, a randomizer function can be activated. This function runs every time the step is activated, so the step's subdivision is constantly being altered. It is then possible to choose the probability of triggering the effect, allowing deterministic control over the effect if the possibility is set to 100%. The subdivision parameter settings are:

1. a second event is triggered 1 step after;
2. a second event is triggered at 1.333 times the tempo;
3. triggers 2 events at 2 times the tempo;
4. triggers 3 events at 3 times the tempo;
5. triggers 4 events at 4 times the tempo;
6. random selection between previous options.

With both parameters set, the effect is placed at the end of the sequencing chain, where if activated it re-triggers the same step with an according number of delayed repeats at the appropriate time measure. There is a second line of steps in the sequencer where the user can get visual feedback about the status of the effect on each step: on or off. As the sequence runs, there is another

toggle, stepDiv, that is activated and deactivated when the effect gets activated and a small slider that shows its subdivision, subDiv.

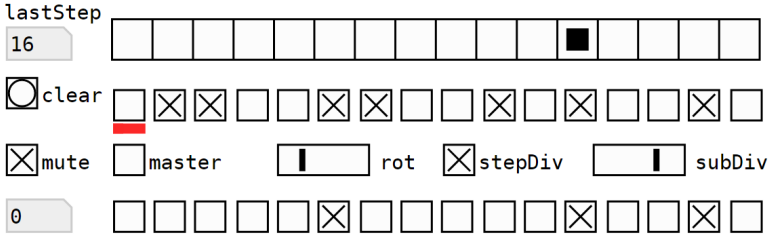


Figure 3.10: Sequencer with visual feedback on Step Divider effect

The Step Divider is also the basis for a Stutter effect. This effect re-triggers the step every sequencer is on at a constant tempo. The re-triggering tempo subdivision can also be controlled by the user using the same subdivision parameter as the Step Divider. It is essentially the same effect set with a 100% probability, but, instead of affecting only the selected instruments, when activated, it works on every sequence playing.

The system's general Reset function, which brings all sequencers to the first step - needed when sequencers running the same amount of steps are dislocated - can also be used as a kind of stutter, but it will always bring every sequence to the beginning.

# Chapter 4

# Application to a Simple Rhythm

This particular sequencing system does not provide a constant output when its sequence and settings are exactly the same as variation is an essential part of its identity. As such, the following examples do not fully represent its behavior. For a more complete understanding, please refer to the accompanying video.

Examples are intended to demonstrate each function by itself. In order to achieve a better demonstration the remaining parameters are left on the default settings, unless otherwise stated. In the end, a general demonstration, where most functions are used, is made.

The order of presentation is:

1. Density
2. Syncopation and Syncopation Variation
3. Indispensability Rankings Rotation
4. Stability
5. Step Divider
6. General Demonstration

$0-pulse is a mere representation of a metronome.

$0-noteOut shows events and their velocities. Each bar represents an event and its height its velocity.

$0-syncopation signals when an event is syncopated.

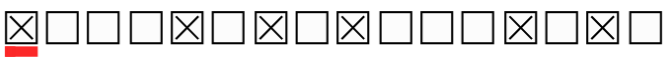

Figure 4.1: Rhythm originally entered by the user

## 4.1 Density

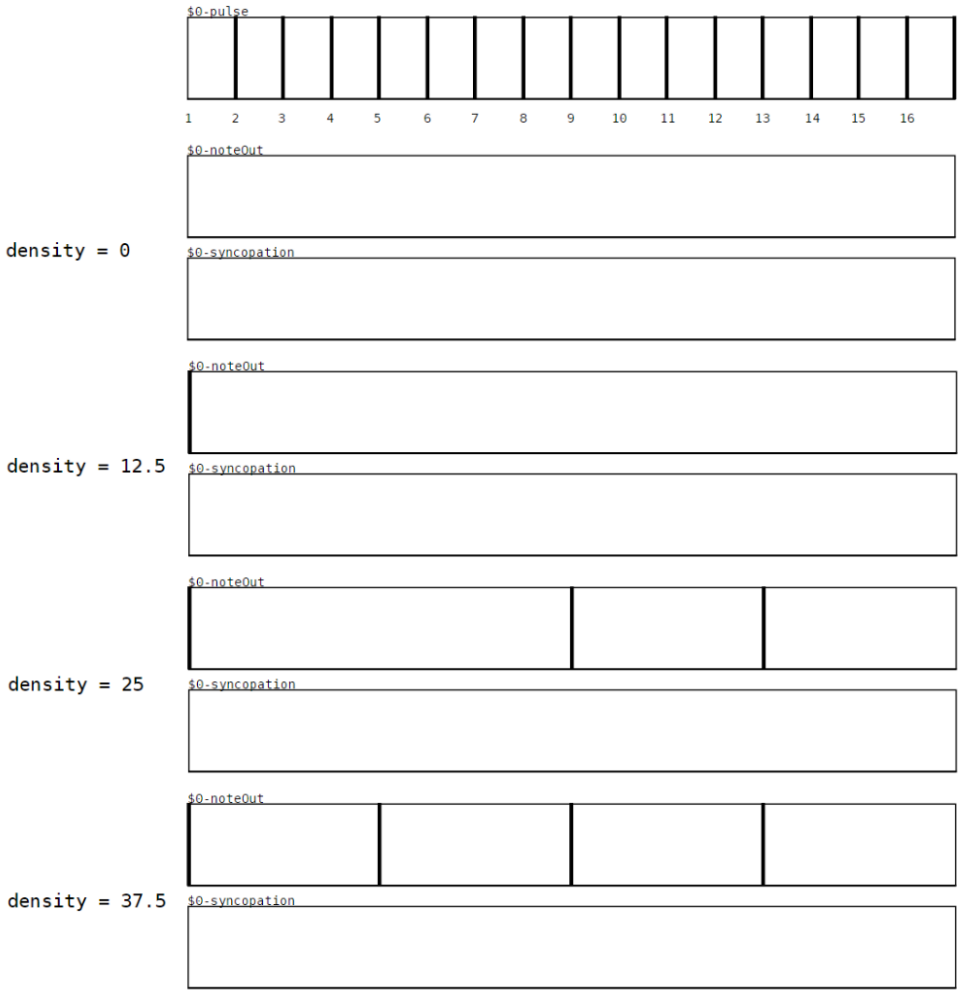


Figure 4.2: Different density settings effect on rhythm (1/2)


Figure 4.3: Different density settings effect on rhythm (2/2)

## 4.2 Syncopation

The first set has no syncopation variation. The second corresponds to an example of how syncopation variation influences the rhythm, syncopation is fixed at 75%. Density and variation are left on their default settings.

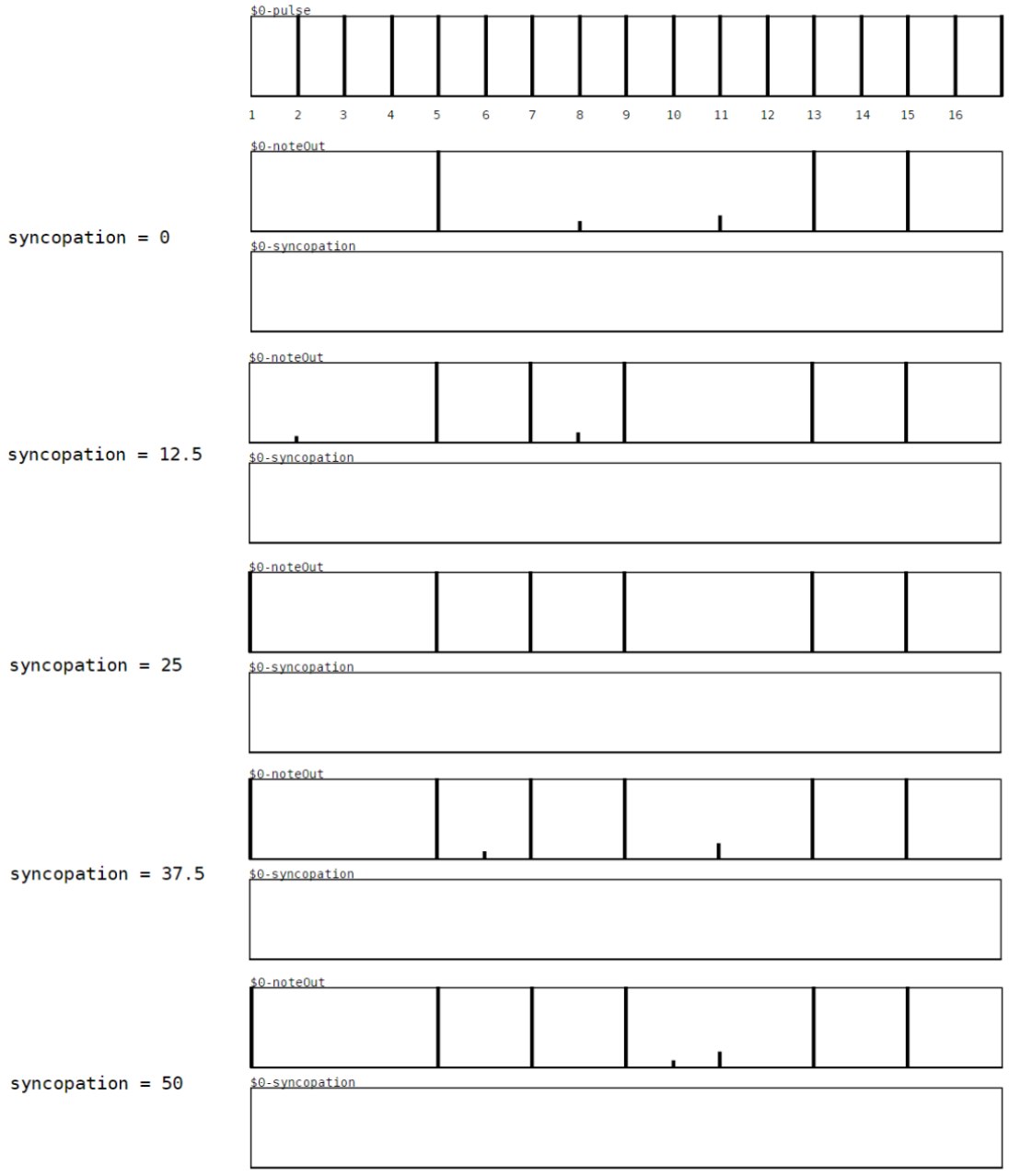


Figure 4.4: Different syncopation settings effect on rhythm (1/2)

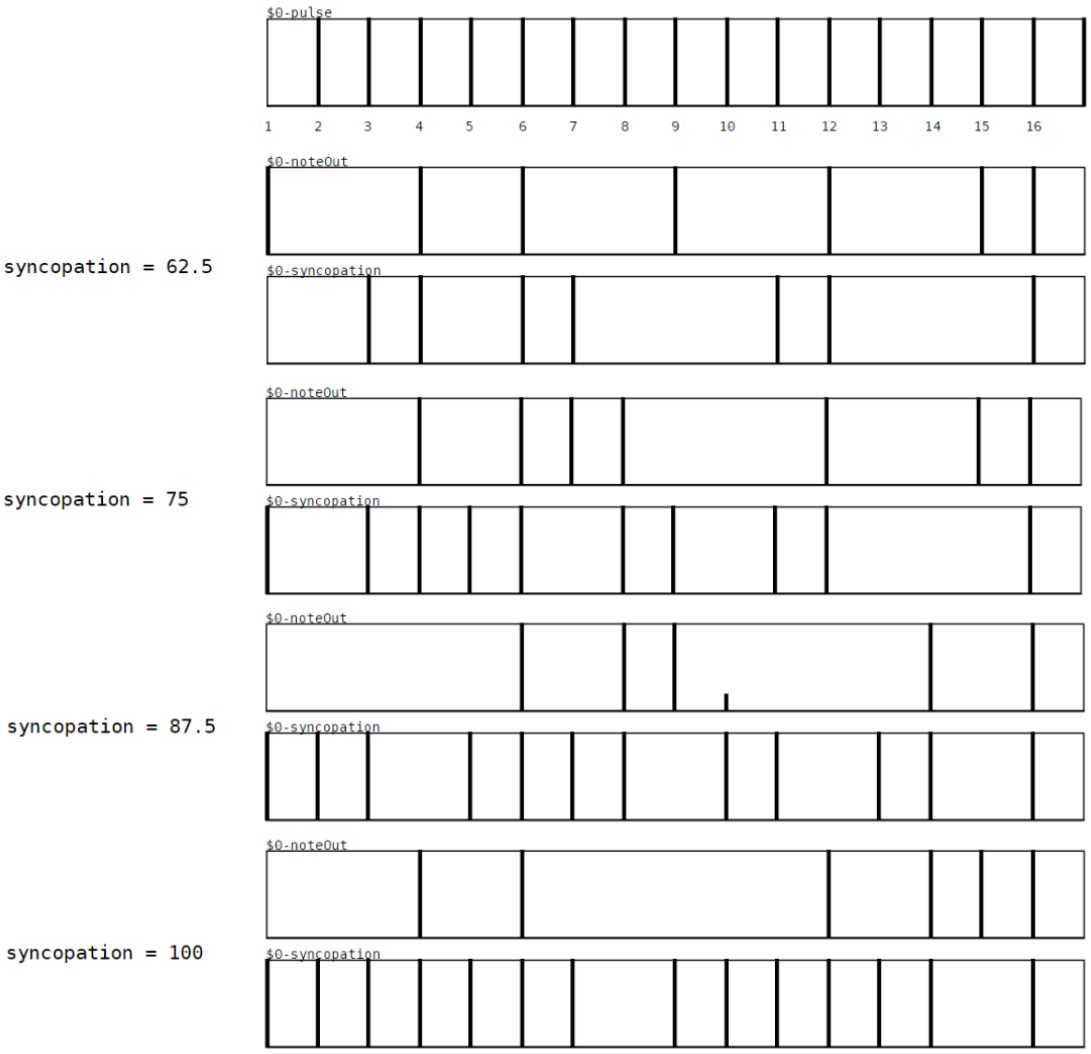


Figure 4.5: Different syncopation settings effect on rhythm (2/2)

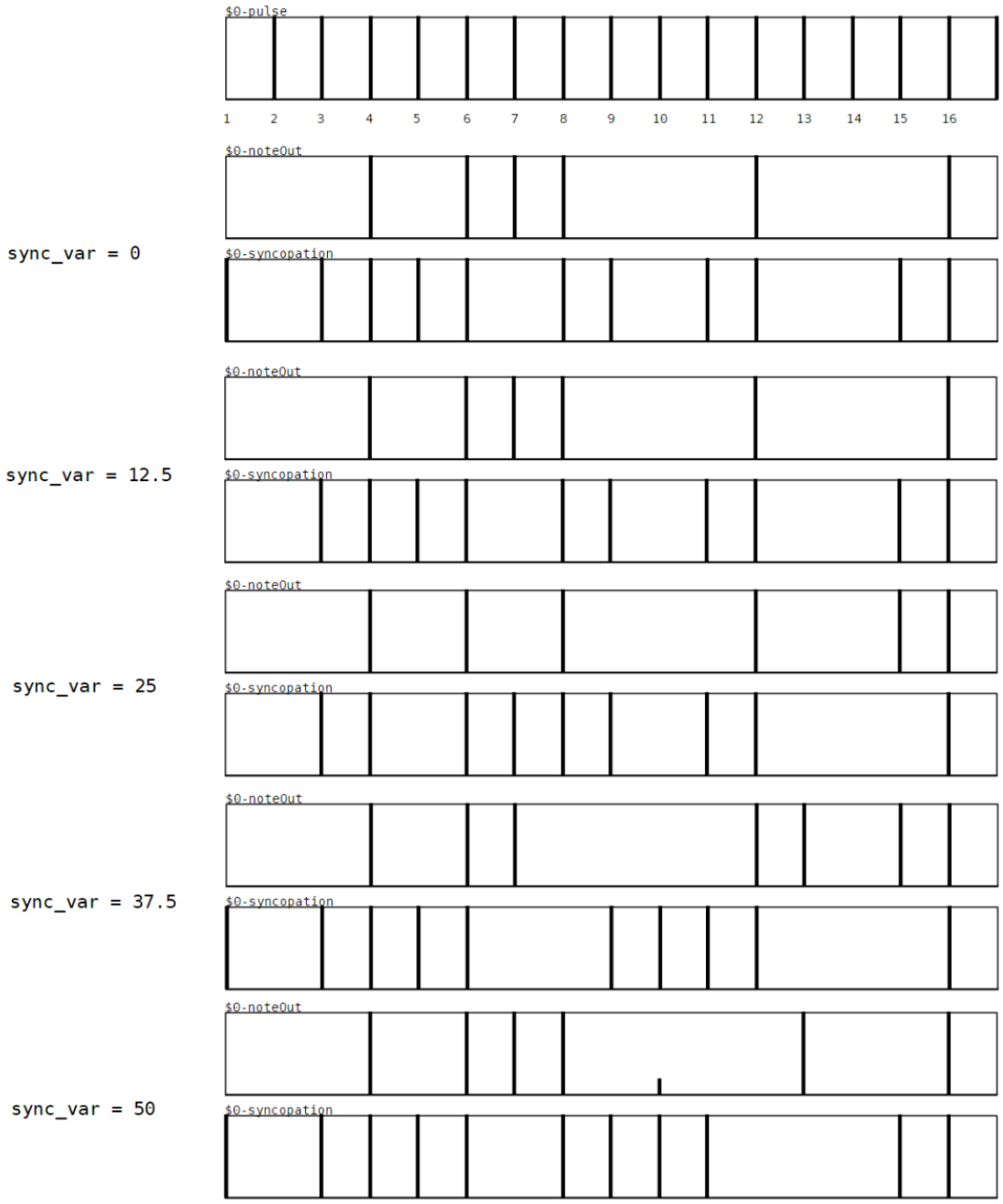


Figure 4.6: Different syncopation variation settings effect on rhythm (1/2)

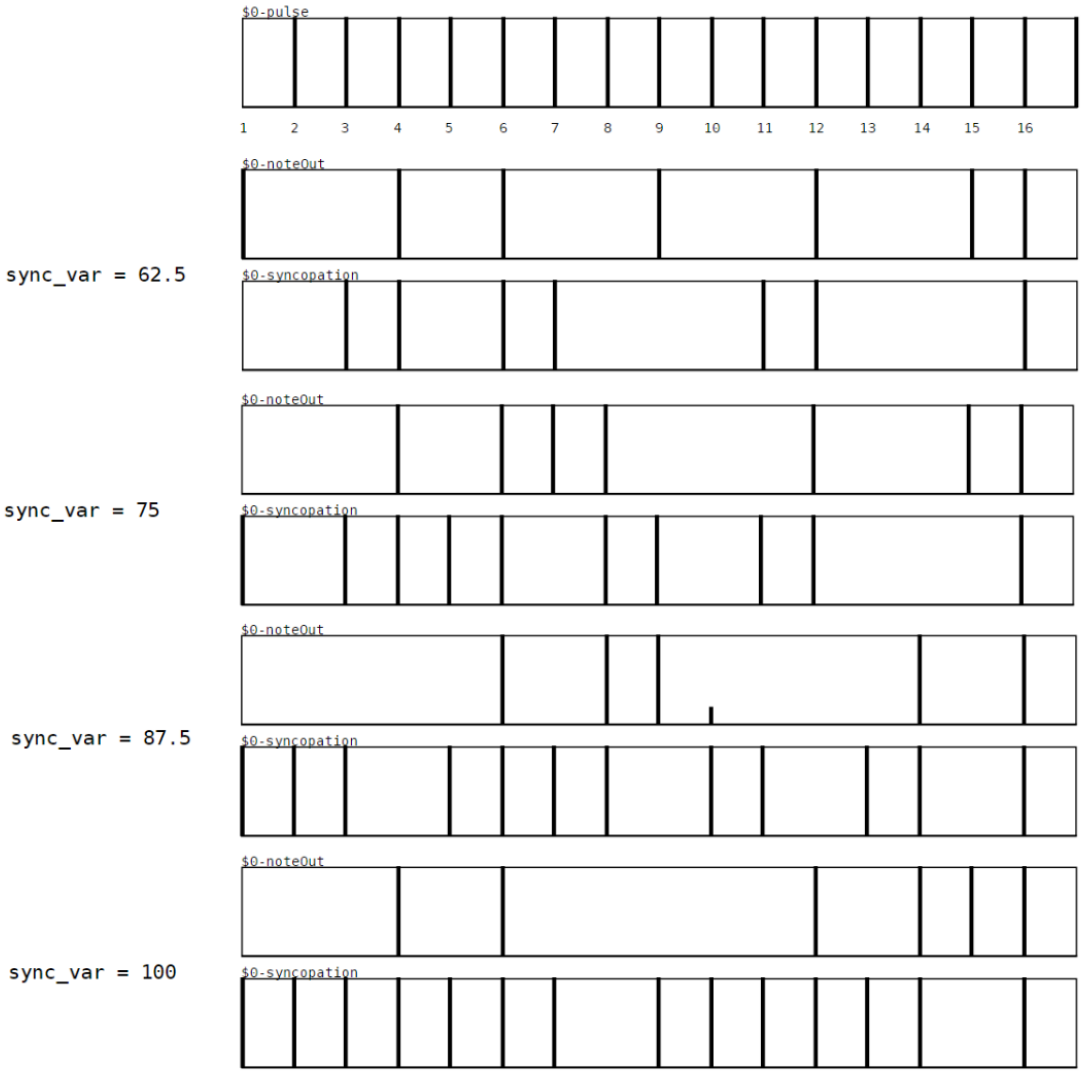


Figure 4.7: Different syncopation variation settings effect on rhythm (2/2)

## 4.3 Indispensability Rankings Rotation

Density is set at 75% for a more clear demonstration of the rotation effect on the rhythm generation around the sequence entered by the user.

Figure 4.8: Indispensability rankings rotation effect on rhythm (1/2)

Figure 4.9: Indispensability rankings rotation effect on rhythm (2/2)

## 4.4 Stability

Density is set at 75% for a more clear demonstration of the stability effect on the rhythm generation around the sequence entered by the user.

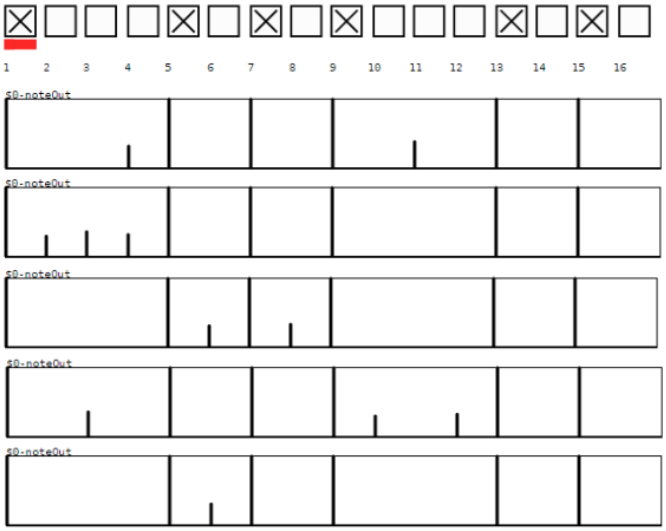


Figure 4.10: Stable rhythm generation

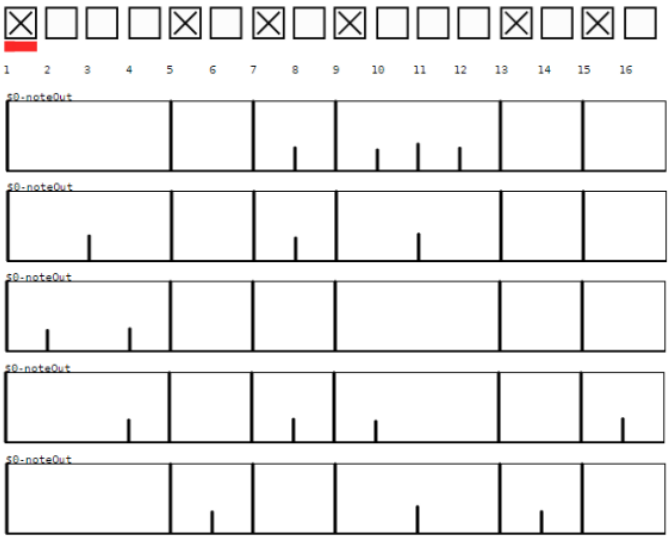


Figure 4.11: Unstable rhythm generation

## 4.5 Step Divider

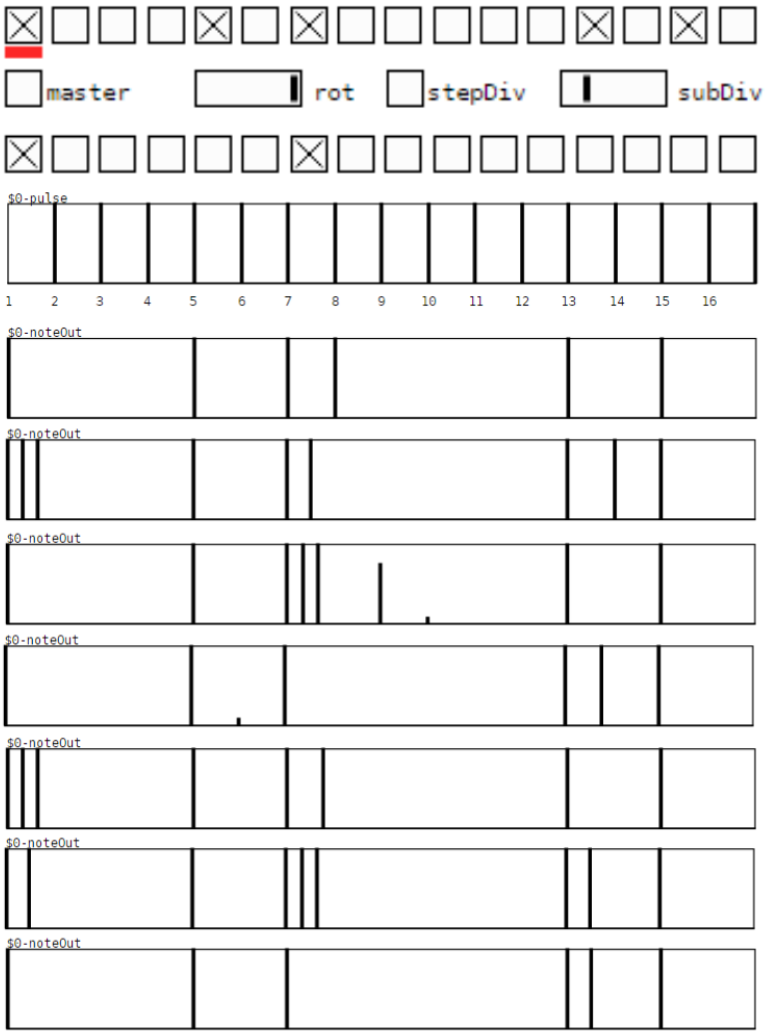


Figure 4.12: Step Divider effect on rhythm

## 4.6 General Demonstration

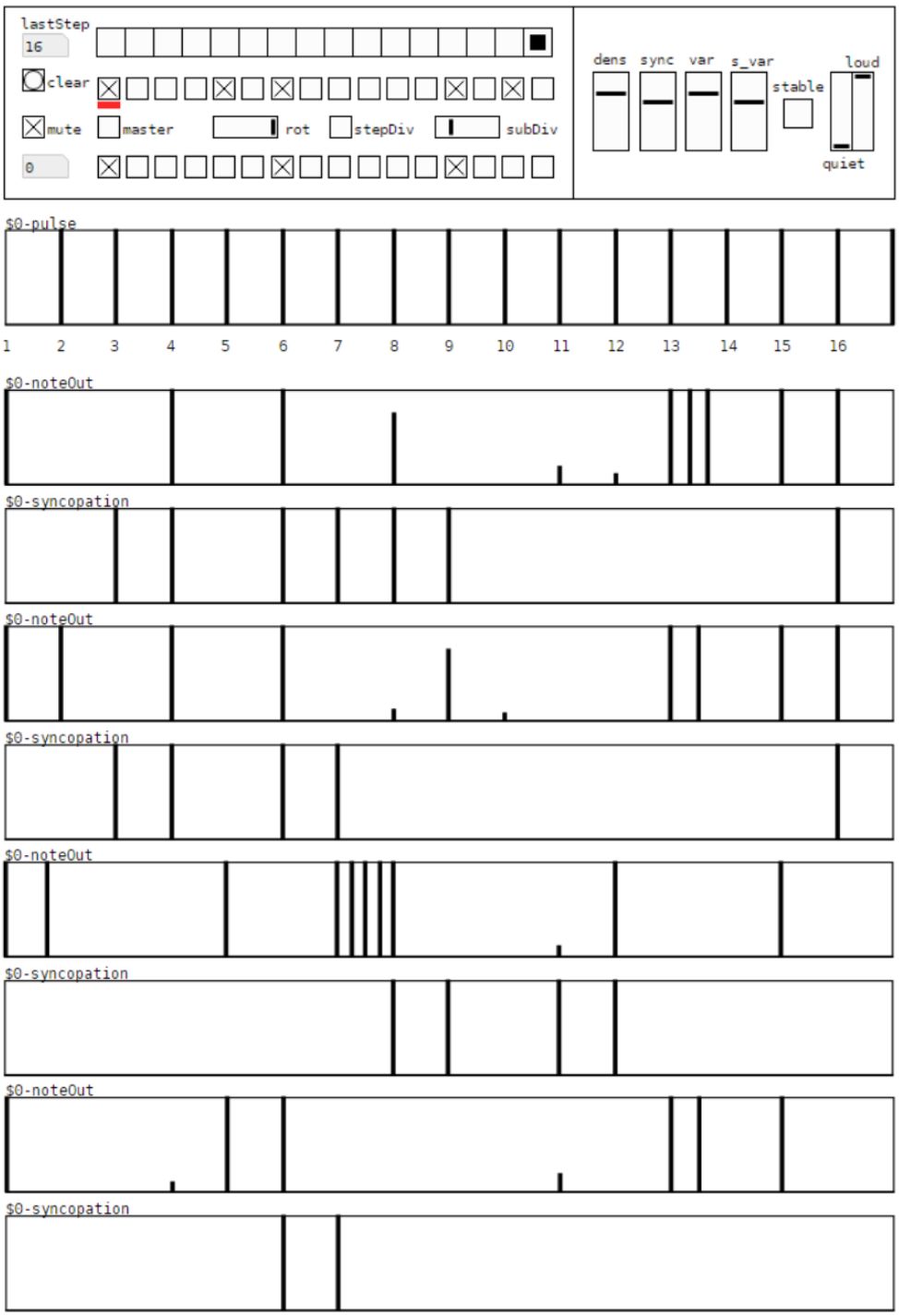


Figure 4.13: Regular use of the sequencer (1/2)

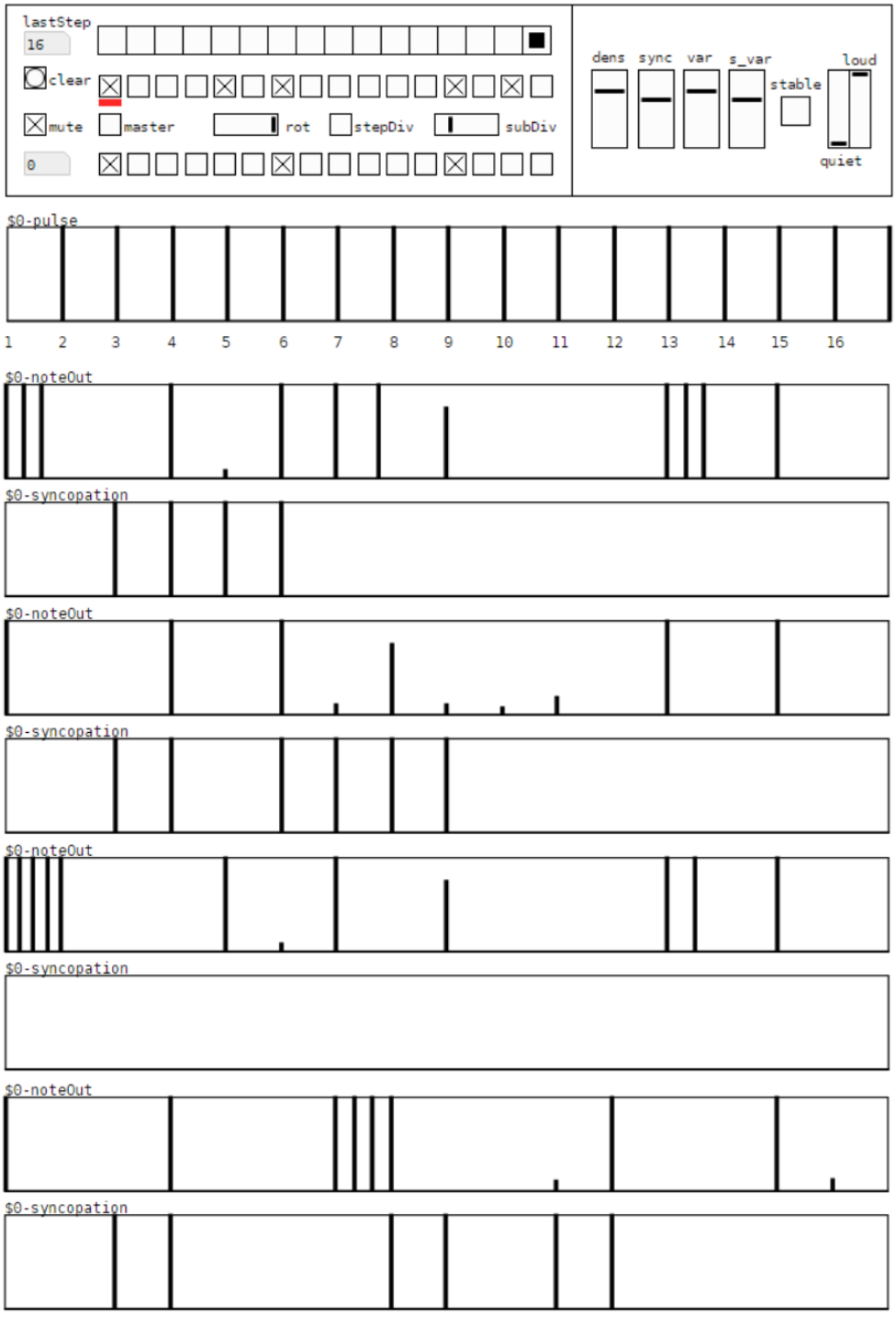


Figure 4.14: Regular use of the sequencer (2/2)

# Chapter 5

# Conclusions and Future Work

This project started from a personal need to have a sequencing system that would be easily programmable to play complex variable rhythms in a performative way. After a review of different techniques available, Georgios Sioros' Rhythmicator was chosen as the generative engine for the possibility of being modified to be integrated in a deterministic style of sequencing while providing a meaningful interactive experience for the user.

From the beginning there was also a concern in developing a system that would be able to be taken away from a computer based setting, as the prototype here presented, to a standalone device. Therefore, the system was developed using open source software Pure Data which makes its integration on mini computers like the Raspberry Pi or BeagleBone, especially with the Bela cape, possible and feasible.

A summary of every step of the work, from the sequence entered by the user to the generative output, is presented next. An analysis of the system is made after. Then, the contribution it represents for the subject is addressed. Finally, future work is discussed.

## 5.1 Summary

This projects consists of a generative monophonic trigger sequencer that can be grouped in multiple instances to form a sequencing system. It works via MIDI, but can, with further work, be adapted to work with OSC or even CV. As mentioned before, the generative engine is based on Georgios Sioros' Rhythmicator and uses his kin_weights and kin_sequencer externals.

The sequencer is divided in 2 separate stages. First the sequence entered by the user is analyzed and then the generative engine generates a rhythm accordingly.

Once the user enters a rhythm on the step-sequencer, the operations performed by the system are the following:

1. Analysis of the amount of steps the sequence has and subsequent attribution of the according indispensability rankings.

2. Correlation of the meter's indispensability values and the sequence entered by the user.

3. In case the amount of steps allows binary and tertiary subdivision, the first and second steps are performed again for tertiary subdivision.

4. The rotation with the best score in the correlation process is selected as the basis for the rhythmic generation.

This process provides the generative engine information about the most appropriate metric feel and the sequence's indispensability rankings rotation

Rhythmicator is an automatic rhythm generator, which is not what is intended in this project. As such, a way was devised to influence how its generation happens, forcing it to output rhythm centered around the sequence entered by the user.

Rhythmicator's engine only needs information about the meter and the stratification level it is supposed to work on. This information is translated into weight values for each pulse. These values are transformed according to the Rhythmicator's density parameter and function as trigger probabilities. Depending on the density value more or less events will be triggered, but the weight values guarantee the original pulse hierarchy of the meter. It was on this level that action was taken to influence the Rhythmicator's rhythmic output. Weight values were subject to a second stage of influence, in which selected steps would have their probability increased and unselected ones would have their probability decreased (with regard to the original hierarchy).

After meddling with the original weight values, the engine is ready to generate variations of the sequence entered by the user according to the density, syncopation, variation, syncopation variation and stability settings.

As the system automatically detects the metric indispensability rankings' rotation, in case more than 1 sequencer is running with the same meter at the same time, it is possible for the first pulse of the meter to not be the same for every sequencer. This may result in conflicting rhythmic generation as the sequence's meters are dislocated from one another.

To prevent this from happening automatically, a command to make 1 of the sequencers act as a master sequencer was introduced. Nevertheless, in case meter dislocation is intended it is possible to overwrite this function and declare a different first pulse for the meter.

Finally, to provide another level of complexity to the resulting rhythms, a probability controlled step divider and a stutter effect were added.

## 5.2 System's Analysis

The process of influencing the original weights of the pulses with the sequence entered by the user proved to be successful. Nevertheless, it creates a big gap between selected and unselected steps which result in a considerable difference in velocity values. This can be corrected using the loud and quiet controls. In fact, this option makes it possible to dial up or down the generated events, providing another level of control over the rhythmic generation.

As the way the Rhythmicator's engine is implemented works with a fixed stratification level for each meter, syncopation does not work on different subdivision levels. When an event is syncopated, it is triggered exactly one step ahead of time. It is still a noticeable effect and produces interesting rhythmic variation, nevertheless, even though different subdivisions can be triggered with the step divider effect, it would be an improvement to have syncopation work on a deeper level. Using the syncopation variation setting, it is possible to make the sequencer syncopate different pulses which also contributes to a more natural feeling of syncopation.

The weight difference between selected and unselected steps not only affects velocity, it also influences the way the engine deals with the metric feel and stabilization. Selected steps make it very hard for the system to generate a rhythm completely detached from the meter's original feel. The same happens for the stability function. When unstable, the rhythm generated should be considerably different at every new cycle which does not happen as the system places selected steps in a much higher hierarchic level.

Rotation of the indispensability rankings is a powerful tool when combined with the velocity range controls. It provides a quick way to generate rhythms with different accentuations.

The step divider is a great tool to generate significant variation in a sequence. By controlling the probability of the effect being triggered and randomizing the subdivision setting, the rhythm generated can be constantly changing while maintaining the same rhythmic characteristics intended by the user.

## 5.3 Contribution of the Work

The sequencing system proposed here is able to deliver complex rhythms by means of a direct and familiar sequencing approach to the user. Even though the lack of an interface capable of providing visual feedback makes the system dependent on the computer screen, its use in a performance environment is straight forward as everything is controlled from the MIDI device.

This project was done with an open-source mentality. Some issues, especially related to the MIDI implementation need further work to be ready for other users, however, when ready, everything will be available online. With the exception of the 2 open source externals developed by

Georgios Sioros, everything was done using Pure Data Vanilla, which makes the further development of the system by other interested parties possible and guarantees its future compatibility with PD to a certain extent.

## 5.4 Future Work

Future work can be done to further develop this sequencing system. The intention to develop a standalone generative rhythm sequencer stands. However, there are also other areas that can benefit from more development:

- Syncopation works in a very limited way. Making the syncopation work on the next stratification level could result in a more interesting effect.
- It would be interesting to have another effect line similar to the step divider that could be programmable to affect a parameter via MIDI CC. It could be used to change the instrument's pitch, sample or length, for example.
- Pattern memories could be introduced to have the possibility of preparing rhythms beforehand for a performance or even to store different rhythms as parts of a song.
- MIDI controllers needs to be easily configured, as it was mentioned before.
- As of now, the system's is controlled by a metro object on Pure Data. Proper transport controls and MIDI tempo information need to be implemented. The possibility of having the system work as a MIDI master or slave to other MIDI devices would also make it better to function with other MIDI sequencers.

For a standalone sequencer to be developed there is also work to be done regarding the interface design, specially regarding how the user gets visual feedback about the settings of each individual sequencer as well as a general overview of the whole system.

# Appendix A

# Implementation

As it was previously stated in the document, all programming took place in Pure Data Vanilla with the exception of the kin_weights and kin_sequencer externals developed by George Sioros for Rhythmicator. Both Pure Data and the Sioros' externals are open-source.

## A.1 Brief Explanation of the Pure Data patch

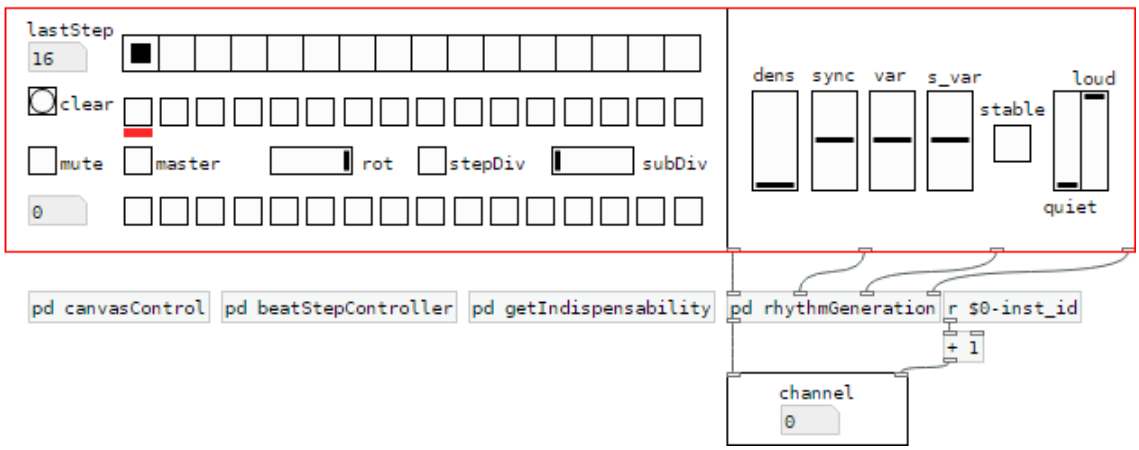


Figure A.1: Sequencer's main patch

Above, in Figure A.1, the main patch of the Sequencer is shown. It provides the user control over everything related to the rhythm generation with visual feedback. On the bottom part it is possible to see the different elements that constitute the system. Tempo and Play/Stop information come from an external main abstraction.

### A.1.1 pd getIndispensability

This sub-patch is responsible for the analysis of the sequence. It is divided in 2 parts. First, on lastStepAnalysis, it checks the amount of steps the sequence has and defines the possible indispensability rankings by getting the possible stratification levels from the number of steps and feeding them to the indispenser abstraction made by Gilberto Bernardes. In case the sequence has only 4 steps, for example, the indispensability ranking is easy to infer, as there is only one stratification option: 3 0 2 1. However, further action is required to understand if a 6 step sequence is of binary or tertiary subdivision: 5 0 3 1 4 2 or 5 0 2 4 1 3. These rankings are stored in a text object and passed on to the second stage.

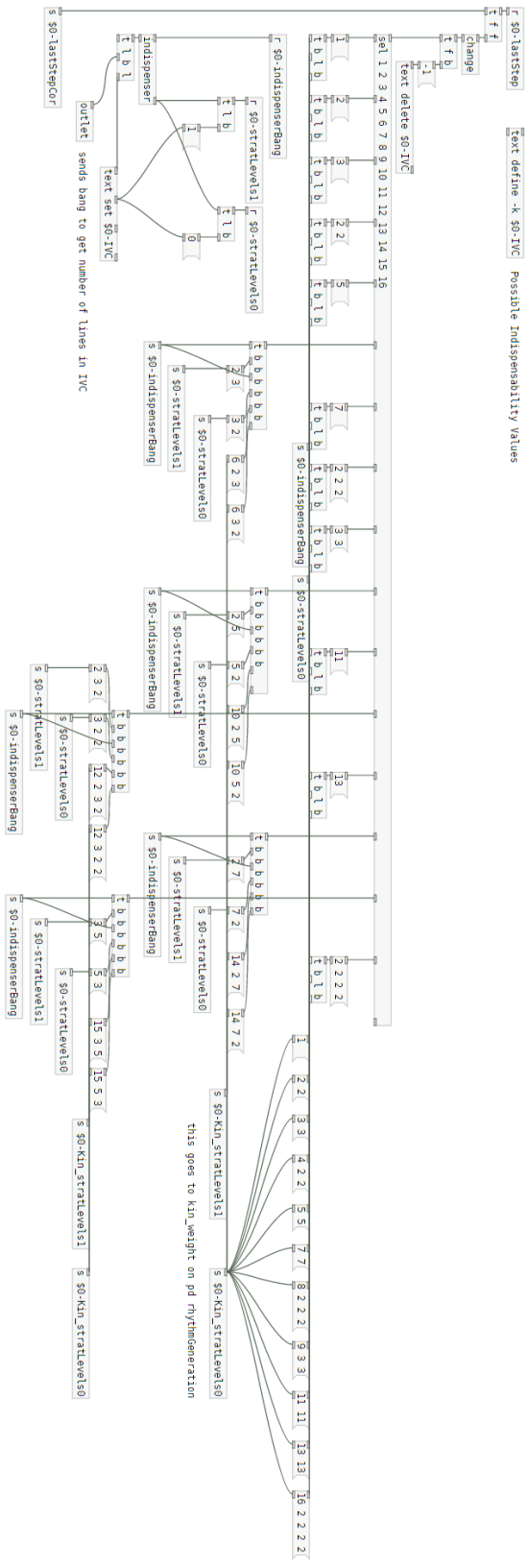


Figure A.2: pd lastStepAnalysis

On the second part of this sub-patch, seqIndispCorrelation, the rankings are compared to all possible rotations of the sequence to see which one is the best match. In case the highest score is shared by more than one rotation it is chosen the one closer to the original first step of the sequencer. The output of this sub-patch consists of the metric indispensability ranking and the chosen rotation that is later applied to kin_weights values and feeds the kin_sequencer generative engine.

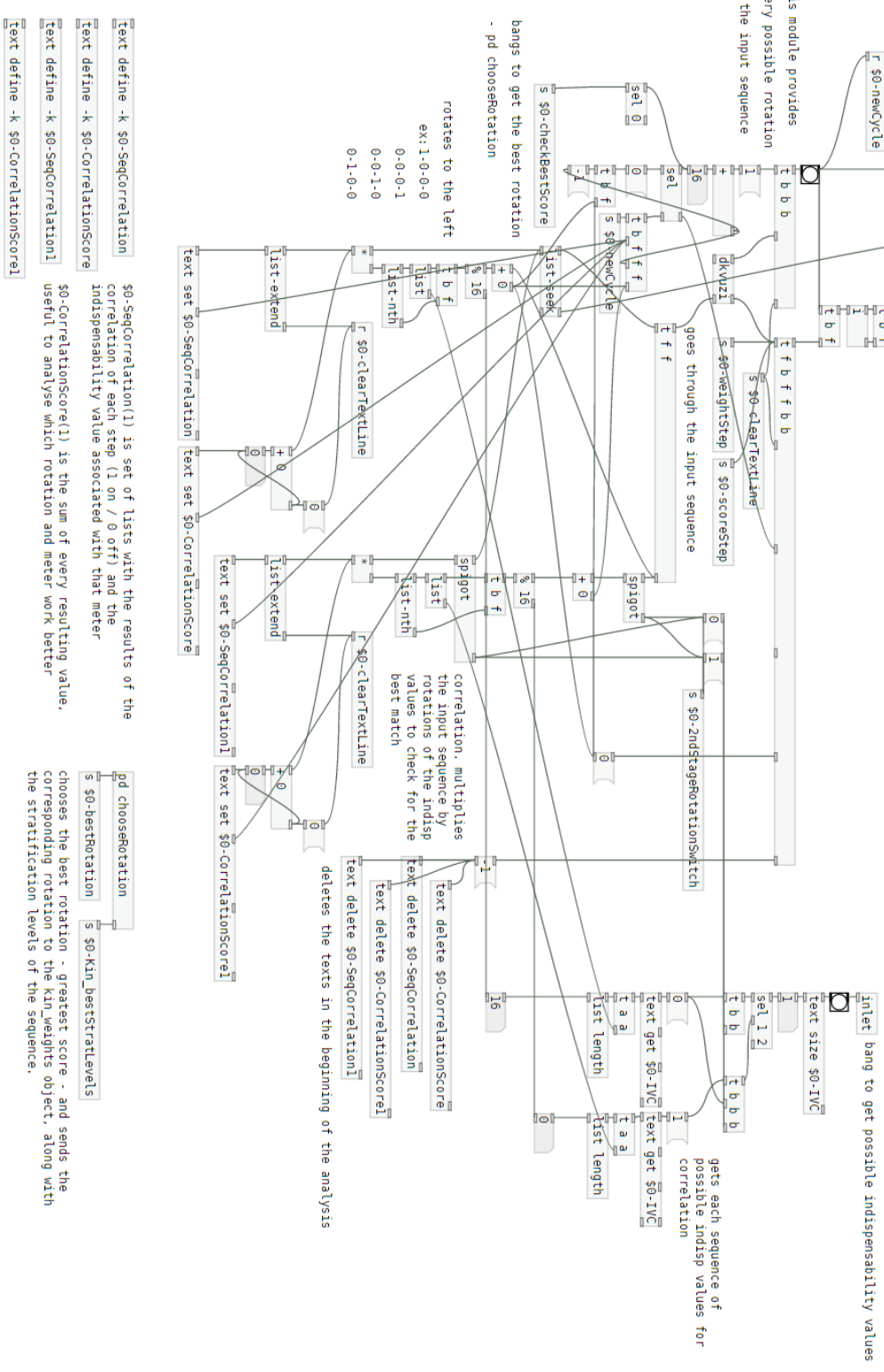


Figure A.3: pd seqIndispCorrelation

### A.1.2 pd rhythmGeneration

The rhythmGeneration sub-patch is where the triggering of events takes place. After receiving the according indispensability ranking, the kin_sequencer object is ready to generate rhythms controlled by the Density, Syncopation and Variation parameters. However, as the kin_sequencer is slaved to the sequence written by the user, it is first required to rotate the indispensability weights based on the information gathered before on the getIndispensability sub-patch. If Rotation is used as an effect, it is on this level that it works.

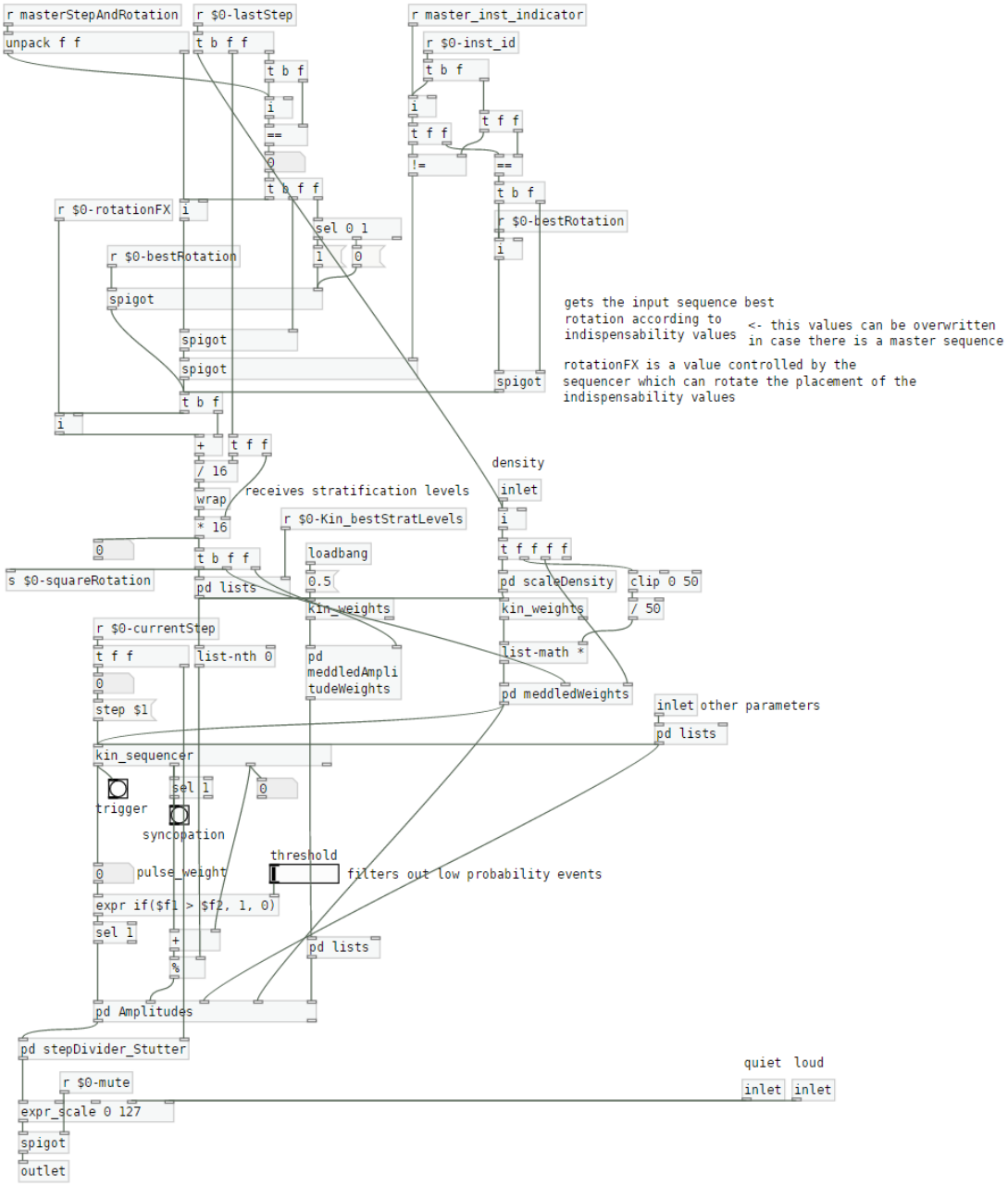


Figure A.4: pd rhythmGeneration

The manipulation of the rhythm generation by the written sequence happens in the meddledWeights abstraction as explained in chapter 3. Amplitude meddling occurs on the meddledAmplitudeWeights abstraction.

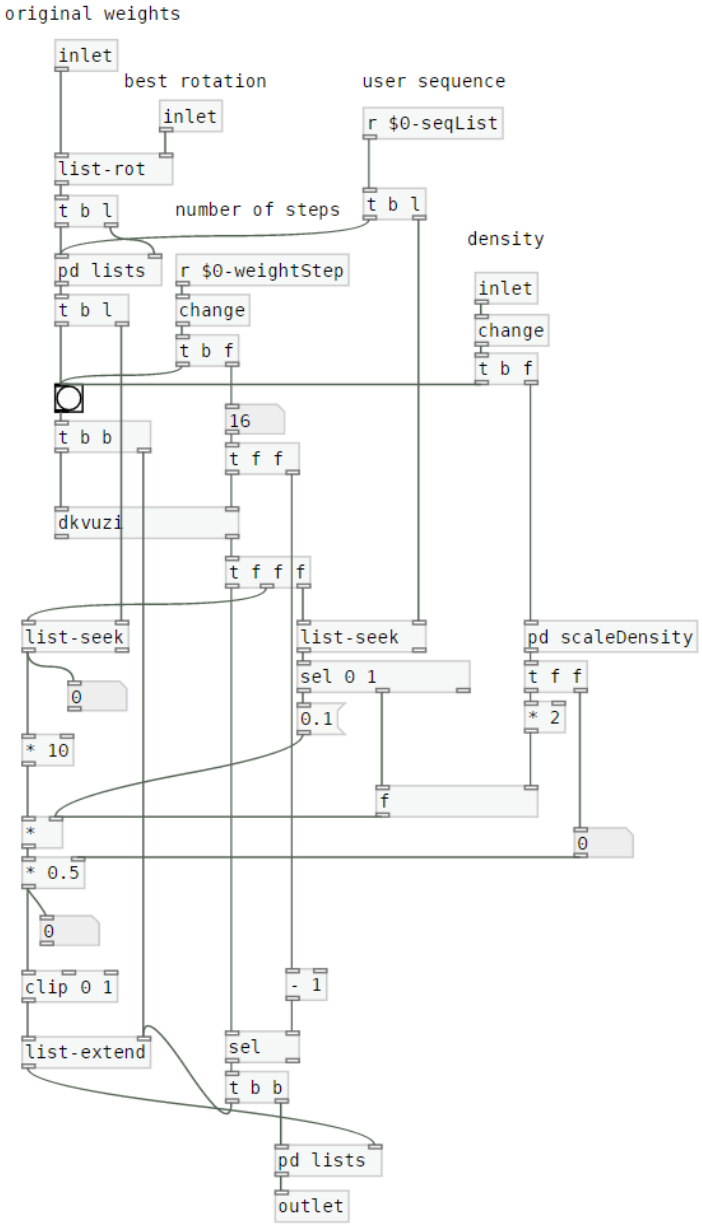


Figure A.5: pd meddledWeights

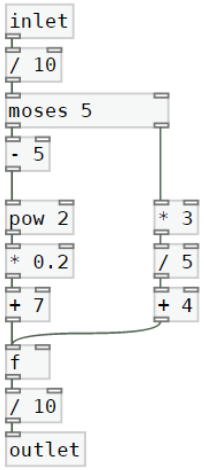


Figure A.6: pd scaleDensity

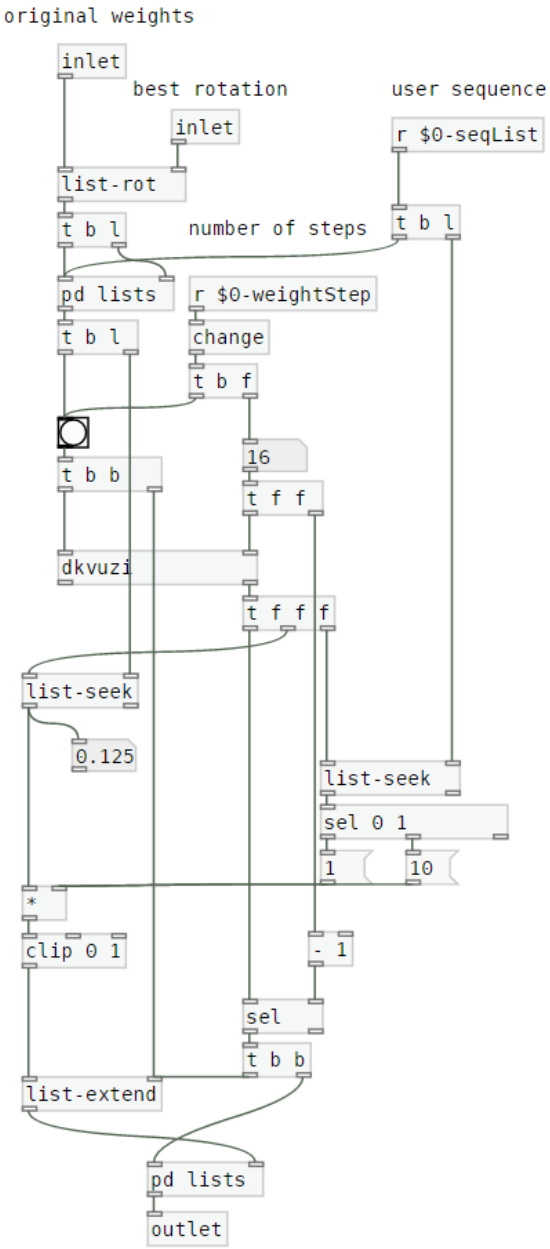


Figure A.7: pd meddledAmplitudeWeights

At the end of the sequencing chain lies the Step Divider and Stutter effect which passes a float value that doubles as a trigger and as the according velocity for the triggered sound. This velocity value is then scaled between the Quiet and Loud limits set by the user.

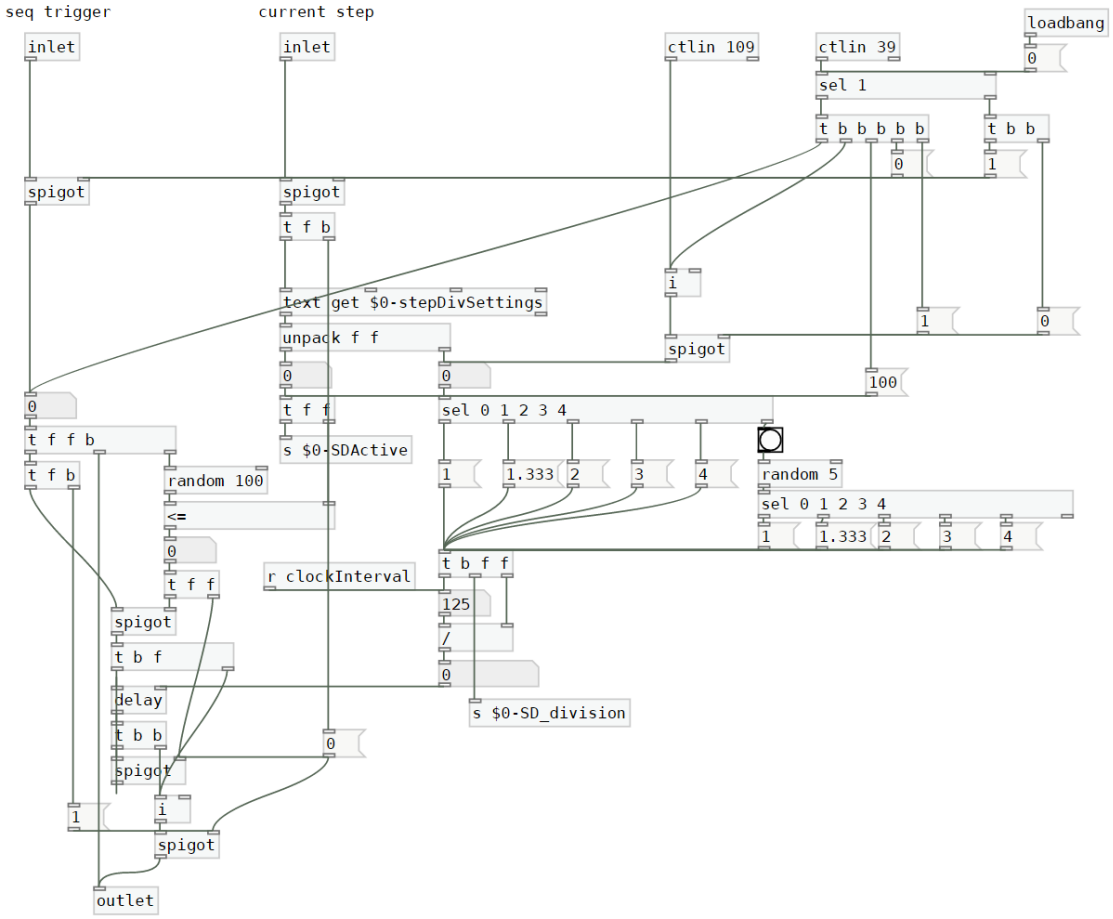


Figure A.8: pd stepDivider_Stutter

Finally, the float number the sequencer outputs has to be translated into a message according to the desired sound module. Each sequencing module is assigned to a MIDI channel and whenever an event is triggered a bang and a velocity message (velocity is sent as MIDI CC #7 to control the volume of the event) are sent through it to play the chosen sound module. This output configuration can be easily configured in a different manner by the user to control different types of devices.

## A.2 MIDI controller configuration

The ideal way to control this sequencing system in a performative manner would be to have a dedicated physical controller. For now, the prototype is controlled by an Arturia Beatstep Pro and the mapping was thought out to have as much "knob per function" as possible.

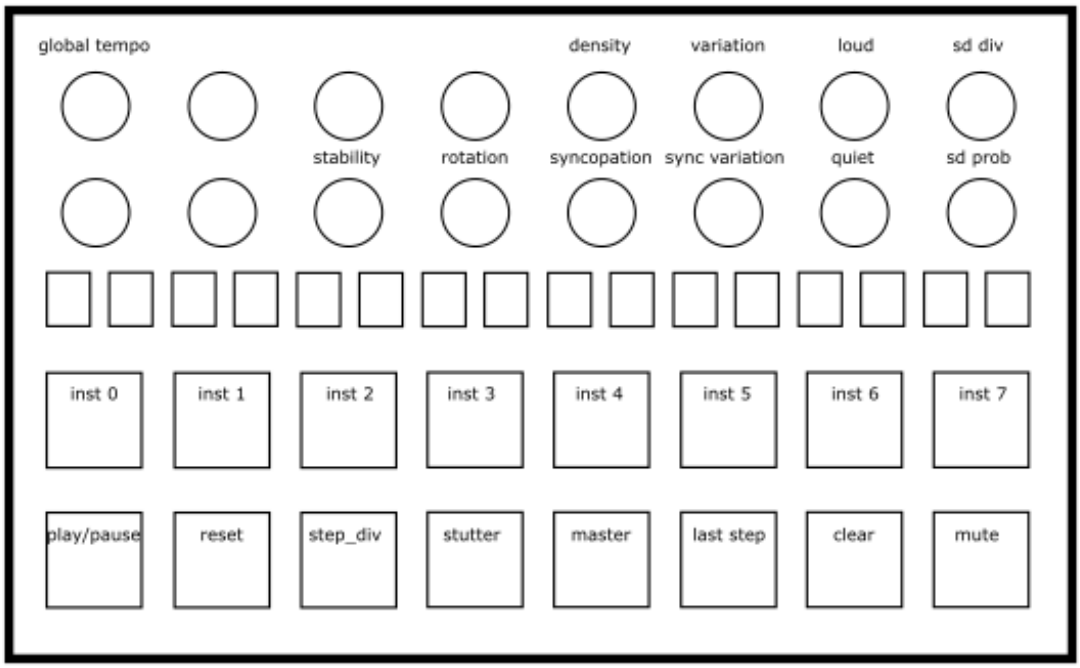


Figure A.9: Arturia Beatstep Pro MIDI mapping

The top pad row selects the instrument(s). By selecting one, or more instruments, the user can enter a rhythm using the 16 buttons. If more than one instrument is selected, the entered steps affect all selected instruments. With the exception of the global tempo and step divider related controls, knobs function in the same way.

Bottom pad row:

- The play/pause button starts and pauses the sequencing system
- Reset makes the sequence go to the first step - this can be used as a stop button after pausing the sequence or it can be used as a rhythmic effect with the sequence running.
- The step divider toggle lets the user access the step divider sequencer. While activated, the user can press a step and choose the subdivision and probability settings for the step divider effect.
- The stutter effect is activated automatically when the pad is pressed and the user can control its subdivision with the step divider subdivision knob.

- The master button makes the active instrument the master.
- By activating the last step pad, it is possible to select what is the last step of the active instruments' sequence.
- The clear button deletes the selected steps on the active instruments. If the step divider pad is activated it deletes the step divider's information.
- The mute button activates or deactivates the selected instruments.

# References


[1] James Andean. Sound and narrative: Acousmatic composition as artistic research. 06 2014.

[2] Curtis Bahn, Tomie Hahn, and Dan Trueman. Physicality and feedback: A focus on the body in the performance of electronic music. 01 2001.

[3] Clarence Barlow and Henning Lohner. Two essays on theory. *Computer Music Journal*, 11(1):44–60, 1987.

[4] Gilberto Bernardes, Carlos Guedes, and Bruce Pennycook. Style emulation of drum patterns by means of evolutionary methods and statistical analysis. 04 2015.

[5] John Biles. Genjam: A genetic algorithm for generating jazz solos. in: Proceedings of the 19th international computer music conference (icmc). pages 131–137, 09 1994.

[6] Kim Cascone. The aesthetics of failure: "post-digital" tendencies in contemporary computer music. *Computer Music J.*, 24(4):12–18, December 2000.

[7] Nick Collins and Alex McLean. Algorave: Live performance of algorithmic electronic dance music. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 355–358, 2014.

[8] Vitor Joaquim Paredes Fernandes. *O papel do laptop performer enquante agente transformador das práticas musicais*. PhD thesis, Universidade Católica Portuguesa, 2015.

[9] W. Tecumseh Fitch. Rhythmic cognition in humans and animals: distinguishing meter and pulse perception. *Frontiers in Systems Neuroscience*, 7:68, 2013.

[10] Francisco Gómez, Andrew Melvin, David Rappaport, Godfried T Toussaint, et al. Mathematical measures of syncopation. In *BRIDGES: Mathematical Connections in Art, Music and Science*, pages 73–84, 2005.

[11] Morgan Jenks. Lr.step, an algorithmic drum sequencer. In *Proceedings of the International Computer Music Conference 2016*, pages 71–73. ICMC, 2016.

[12] Alexader R. Jensenius and Michael J. Lyons, editors. *A NIME Reader: Fifteen Years of New Interfaces for Musical Expression*, volume 3. Springer International Publishing, 2017.

[13] Alex McLean. Angry - /usr/bin/bash as a performative tool, 2003. Accessed: 2018-06-25.

[14] Curtis Roads. *Composing Electronic Music: A New Aesthetic*. Oxford University Press, 2015.

[15] Georgios Sioros. *Syncopation as Transformation*. PhD thesis, Faculdade de Engenharia da Universidade do Porto, 2015.

[16] Georgios Sioros and Carlos Guedes. A formal approach for high-level automatic rhythm generation. In Reza Sarhangi and Carlo H. Séquin, editors, *Proceedings of Bridges 2011: Mathematics, Music, Art, Architecture, Culture*, pages 233–240. Tessellations Publishing, 2011.

[17] Godfried T. Toussaint. Generating "good" musical rhythms algorithmically. 2010.

[18] Owen Vallis and Ajay Kapur. Community-based design: The democratization of musical interface construction. *Leonardo Music Journal*, -(21):29–34, 2011.