Exploring Polyrhythms, Polymeters, and Polytempi with the Universal Grid Sequencer framework

SAMUEL J. HUNT, Creative Technologies Laboratory

Fig. 1. Large format grid controller, made from 4 smaller grid controllers

Polyrhythms, Polymeters, and Polytempo are compositional techniques that describe pulses which are desynchronous between two or more sequences of music. Digital systems permit the sequencing of notes to a near-infinite degree of resolution, permitting an exponential number of complex rhythmic attributes in the music. Exploring such techniques within existing popular music sequencing software and notations can be challenging to generally work with and notate effectively. Step sequencers provide a simple and effective interface for exploring any arbitrary division of time into an even number of steps, with such interfaces easily expressible on grid based music controllers.

The paper therefore has two differing but related outputs. Firstly, to demonstrate a framework for working with multiple physical grid controllers forming a larger unified grid, and provide a consolidated set of tools for programming music instruments for it. Secondly, to demonstrate how such a system provides a low-entry threshold for exploring Polyrhytms, Polymeters and Polytempo relationships using desynchronised step sequencers.

CCS Concepts: • **Human-centered computing** → User interface programming; • **Applied computing** → **Sound and music computing**.

Author's address: Samuel J. Hunt, Creative Technologies Laboratory, UWE Bristol, Samuel.hunt@uwe.ac.uk.

2020. Manuscript submitted to ACM

Additional Key Words and Phrases: Polyrhythm, Polymeter, Polytempo, Grid Controllers

ACM Reference Format:
Samuel J. Hunt. 2020. Exploring Polyrhythms, Polymeters, and Polytempi with the Universal Grid Sequencer framework. 1, 1 (November 2020), 11 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Linear sequencer workflows generally enforce synchronization between tracks, whereby it is not possible to have more than one tempo or meter in concurrency [10]. For example one track at 120 bpm and another at 130, or one track in 4/4 and another in 9/8. Different types of notation allow or inhibit this in different ways. For example western score notation makes compound time permissible with different voices appearing on a single stave, or various compound notation across different staves. A piano roll notation allows virtually any discrete time division but can require tenacity and fine motor control to drag a note into a position. Step sequencer notation provides a simple way of expressing rhythms by dividing a period of time evenly into a fixed number of intervals (steps).

A step sequencer generally divides a sequence into evenly spaced steps, whereby a step corresponds to an individual note that is either on or off at that position [11]. Steps run left-to-right and voices are stacked vertically. Step sequencers, unlike traditional notation, make it easy to divide a bar of music into an arbitrary number of steps, for example dividing a bar of 4/4 into 11 identical pulses. Step sequencers have some novel properties, for example each row can be mapped; linearly, to a given scale, or to an arbitrary note value (drum sequencer). Step sequencers exist in both physical hardware and in software. We propose a simple step sequencer interface using physical hardware for exploring time, whereby we can have multiple step sequencers running together to both illustrate and audibly compare the effects of Polyrhythms, Polymeters, and Polytempi.

Grid controllers are physical interaction devices made up of a matrix of backlit buttons (see Figure 1). The interface provided by a grid controller lends itself well for representing and interacting with step sequencers. Many such controllers exist (explored in the next section) however the tools and frameworks for designing sequencers and more general purpose instruments for grid controllers are somewhat lacking. To enable us to explore time for the purpose of this project we created a general purpose framework for enabling the design of instruments for grid controllers (section 3). From this we demonstrate a step sequencer based approach using a large format grid controller for exploring complex rhythms (section 4). The large grid controller discussed in this project was built several years prior to this project and more details about its origin can be found here[1] .

2 BACKGROUND

2.1 Polyrhythms, Polymeter, and Polytempo

A polyrhythm is the simultaneous use of two or more rhythms that are not readily perceived as deriving from one another [1]. Given 2 bars of the same length and tempo (stacked vertically), a polyrhythm would be dividing the upper bar into 4 beats and the lower one into 3 beats. These beats would be in sync on the first beat and drift or phase for remainder of the bar (figure 2). A polymeter is where two sequences are played using different meters, but with the same tempo. For example a pattern that repeats a sequence over 5/4 played against one repeating over 4/4 (figure 3). After 20 pulses the original patterns will repeat again (the lowest common denominator of two time signatures). Polyrhythm and polymeter differ in that the first repeats every measure, and the latter at the phrase level [5].

[1] http://launchpadmegamini.blogspot.com/

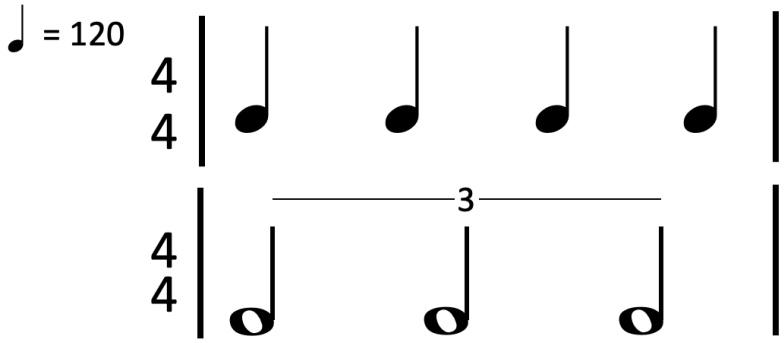

Fig. 2. 4/3 polyrhythm.

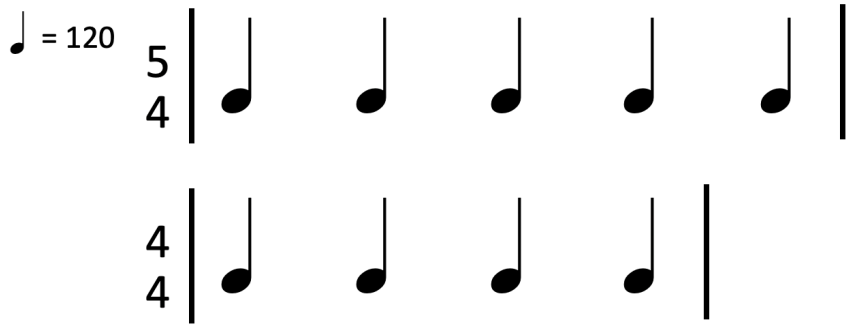

Fig. 3. 5/4 and 4/4 polymeter

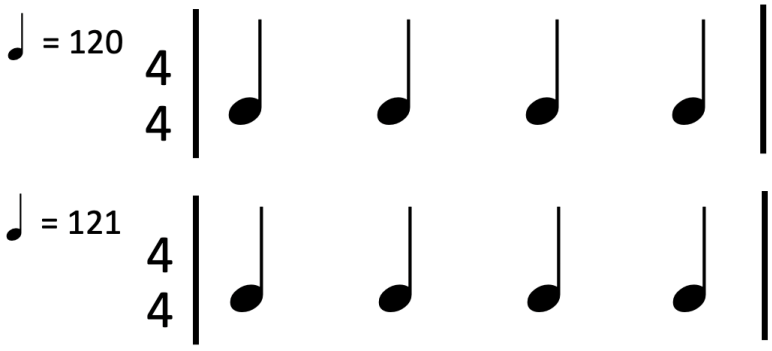

Fig. 4. two sequences using polytempo

Polytempo is whereby two sequences are played at different tempos, the point at which these sequences repeat can be significantly longer than sequences employing either polyrhythm or polymeter. For example a 1 bar repeating phrase sequence played at 120 bpm and at 121 bpm simultaneously will take 4 minutes to repeat (figure 4).

Renney and Gaster [10] state traditional notation software struggles to represent more complex manipulations of musical time. Additionally, expressing two time signatures in parallel (polymeter) or working with concurrent time (polyrhythm) is not generally well supported in notation software and thus requires undesirable 'work-arounds'. However some existing software systems do support these compositional methods.

Nash's [9] Manhattan tracker software enables a user to work with complex rhythms by subdividing rows in the tracker into a number of subdivisions. For example taking a row with a resolution of 1/32 note and dividing this into 7 smaller divisions, this effect can be repeated between tracks creating complex rhythmic structure. Malmberg's IRIS software sequencer [7], represents a step sequencer around the edge of a circle. The system supports having multiple sequencers within the circle so that polyrhytmic music can be explored. Renney and Gaster's [10] work presents a novel physical interface for exploring polyrhythms and polymeters on a 3D printed circular interface. Dorin [3] has explored polyrhythms in a more abstract form using cellular automata. More general purpose programming languages such as Supercollider [12] would allow a user to program any number of complex rhythms but requires competency of programming for an end-user. More formal methods for composing with complex rhythms is given by Dobrian [2].

2.2 Grid Controllers

Grid controllers are generally defined as a matrix of buttons that output button state information when pushed. Many of these controllers have LED lighting behind each button so that visual feedback can be displayed. The first commercial example of a grid controller is commonly considered the Monome [4]. More modern controllers such as the Launchpad Pro[2] and Ableton Push [3] use velocity and pressure sensitive pads with full RGB feedback. Unlike traditional keyboards, grid controllers are more general purpose with specific software providing the instrumental interface and mapping for musical applications. Other than the Monome 256 (a 16 x 16 grid controller), Yamaha Tenori-on[4], and Polyend SEQ[5], larger grid controllers are rare and notably expensive.

Many purpose built musical applications exist for grid controllers, utlising Max for live patches, custom firmware, or programmed in a general purpose language. For example, Monome has a community of developers creating applications written using Max [8]. Novation's launchpad pro has the ability to write programs and embedded them directly on the device [6]. Ableton's push controllers provide specific interfaces when interfaced with Ableton Live. No general purpose framework exists for building instruments on a grid controller, but rather each device has its own set of tools and requirements. For example, a purpose built application for a Launchpad written in Max would require a significant rewrite to enable interaction with a push controller. Given the amount of common functionality between grid controllers, a framework that enables instruments to be designed for a *'general purpose grid controller'* allows developers and musicians to more easily develop and share instruments.

3 UNIVERSAL GRID SEQUENCER

The Universal Grid Sequencer (UGS) is an open source C++ framework that provides a series of musical applications for grid based controllers developed for this project. There are a number of components later discussed that make the system universal. But essentially it is a software package that allows multiple grid based controllers to be connected together as if they were one large grid controller and then position various sequencers and musical instruments on it. An instrument written specifically for one grid controller can easily be made to work with another. We briefly discuss

[2]https://novationmusic.com/en/launch/launchpad-pro
[3]https://www.ableton.com/en/push/
[4]https://www.yamaha.com/en/about/design/synapses/id_005/
[5]https://polyend.com/seq-midi-sequencer/

```
// [1] the inner set is size {16 steps, 4 voices}, the outer set is position {x: 1, y:1}
OGController * drumControler = new OGControllerSequencerSimple( {16, 4}, {1,1});

//[2] now we setup the note map, each index is the voice (row)
drumControler->getNoteMap().values[0] = 36; //kick
drumControler->getNoteMap().values[1] = 38; //snare
drumControler->getNoteMap().values[2] = 42; //closed hi-hat
drumControler->getNoteMap().values[3] = 46; //open hi-hat

//[3] add our new controller to the session
session->addNewController(drumControler);

//[4] create a new clock with a tempo of 140 bpm
mClock.addNewClock(140);

//[5] we only have 1 clock and 1 controller so we assume 0 index
mClock.getClock(0)->addController(session->controllerForIndex(0));

//[6] start the master clock
mClock.start();
```

Fig. 5. Code example for setting up a step sequencer. See AMSequencerSimpleDrumSequencer in the source code.

the core concepts as a full detailed description of the system is beyond the scope of the project, full source code is available from[6].

Firstly all MIDI devices are connected through a device object. The device object is responsible for providing the mapping between a device's physical control mapping and the virtual logical mapping employed by the system. For example, the Launchpad controller used by this project sends note on messages for each pad. Each row is represent by increments of 16 and each column by 1. These are translated by the device component into an XY coordinate position. The physical device itself can be oriented in one of four predefined positions (rotated in 90 degree increments) with the software automatically computing the transformations. The device object must also provide the reverse mapping, i.e. given a XY coordinate which pad does this relate back to. We propose that any grid based MIDI controller can be used with the system providing a device and its mapping is defined. For practical reasons we have not tested every variant of grid controllers. The basic requirements for any device is that it supports a state based button press (sending on and off messages), has LED feedback under each button and communicates via MIDI or OSC.

LED feedback is a core part of many grid based controllers, and decoupled from a pad's input. The device object also deals with mapping an arbitrary RGB colour value into a device's physical ability to represent this. Many controllers (Push, Launchpad Pro) support full RGB colour, however some controllers (the Launchpad used by this project) support only red and green with a 2 bit colour depth for each channel. In this instance the red and green components are down-sampled to be compatible.

The device manager is responsible for connecting and positioning several devices together to build an overall grid. The position, size and orientation of these devices can be arbitrary. For example, in this project we have four launchpad devices positioned in a 2 by 2 grid, with each device rotated through 90 degrees (see figure 1). At this point we have an overall grid size of 18 by 18. The device manager is responsible for translating messages from and back to individual devices. For example a message from the bottom right launchpad at its local position 0,0 becomes 9,9 in the overall grid. Likewise, sending LED feedback to position 9,9 will return that feedback to the bottom right launchpad. At this point input messages are transferred from the MIDI thread into the master clock thread using a circular buffer.

Furthermore, all the IO between devices is setup and focus can now be placed on designing controllers. A controller is simply a musical instrument or sequencer of arbitrary size and position within the grid. A session object manages an arbitrary number of controllers, and transfers messages from the device manager onto the relevant controller. For

[6]https://github.com/Sjhunt93/Universal-Grid-Sequencer

example we might have 3 controllers, a drum pad on the top right and note scaler on the top left and a 16 step sequencer occupying the bottom half of our grid. Controllers generally fit into one of 3 categories, momentary controllers, sequencer controllers, and setting controllers. The first of these is the most simple and generally maps input into output MIDI messages. For example we could have a 4 by 4 drum pad that maps each XY position to a MIDI number corresponding to specific drum sounds (e.g. kick = MIDI note 36, snare = MIDI note 38). Sequencer controllers are similar, but instead output notes at some point in the future and resemble traditional step sequencers. These respond to messages sent from clock objects that provide timing information (discussed shortly). Setting controllers send messages that control the state of other controllers, for example switching scales, transposing, selecting sequences, and others.

The master clock is responsible for transferring incoming messages into the individual controllers (through the session manager) and sending LED feedback to the physical devices, but also keeps track of and manages a number of clock objects. Clock objects send regularly pulsed messages to a given number of controllers. There is a 1-to-many mapping between clocks and controllers, however for simplicity 1 clock might control every controller ensuring synchronicity between them. For more complex examples we might have 4, 16-step sequencers each with 4 voices on our 16 by 16 grid, with each sequencer receiving a different clock. This allows each sequencer to go in and out of phase and sync with each other.

All LED feedback is sent to a LFXBuffer (LED effects buffer). Each controller has its own buffer. For example a controller with a size of 5 by 5 has a buffer of the same size. The device manager holds a master buffer that is the same size as the grid. Around 60 times a second the buffers from each controller are transferred to the master buffer and positioned accordingly. The master buffer is a representation of the physical device so therefore output messages are only sent if the state has changed which avoids overloading the communication output stream.

The system is entirely written in C++ and currently configured through changing and modifying source code which is relatively simple to do. Further iterations of the project intend to provide a drag and drop user interface and/or a domain specific language for setting up controllers. Figure 5 shows some example code for creating a step sequencer, and Figure 6 shows the output of this on the physical grid. To create a simple step sequencer we would allocate an instance of it (step 1), and also supply the two required setup parameters: a size (which for this controller determines its number of steps and voices) and a position on the grid. We then set the note numbers for each row (step 2), and add it to the session (step 3). As the controller is a sequencer type (see previous section) we need to create a clock and register this controller to it (steps 4 and 5). If we wanted to add a second sequencer we would repeat the above process. The sequences could either share the same clock or alternatively register a new one. We finally start the master clock (step 6). All MIDI output messages from the program are sent out via a virtual MIDI device and synthesised in an external program, these messages are separate to the LED Feedback (also sent as MIDI) messages sent back to the physical devices.

4 EXAMPLES

Whereas the previous section presented the system as a generalised tool for working with grid based controllers, this section explores how a large grid based controller easily supports experimenting with time using examples in polyrhythms, polymeters, and polytempi. A large grid enables us to have multiple sequences not only running at once but remain visible.

Although creating an arbitrary number of virtual step-sequences in software (for example in Max) is easily possible, physical hardware provides tactile and sensory feedback. In the examples below, 16 steps corresponds to a bar of music in 4/4 time. All example code is available from the open source repository, the accompanying video demonstrates each of

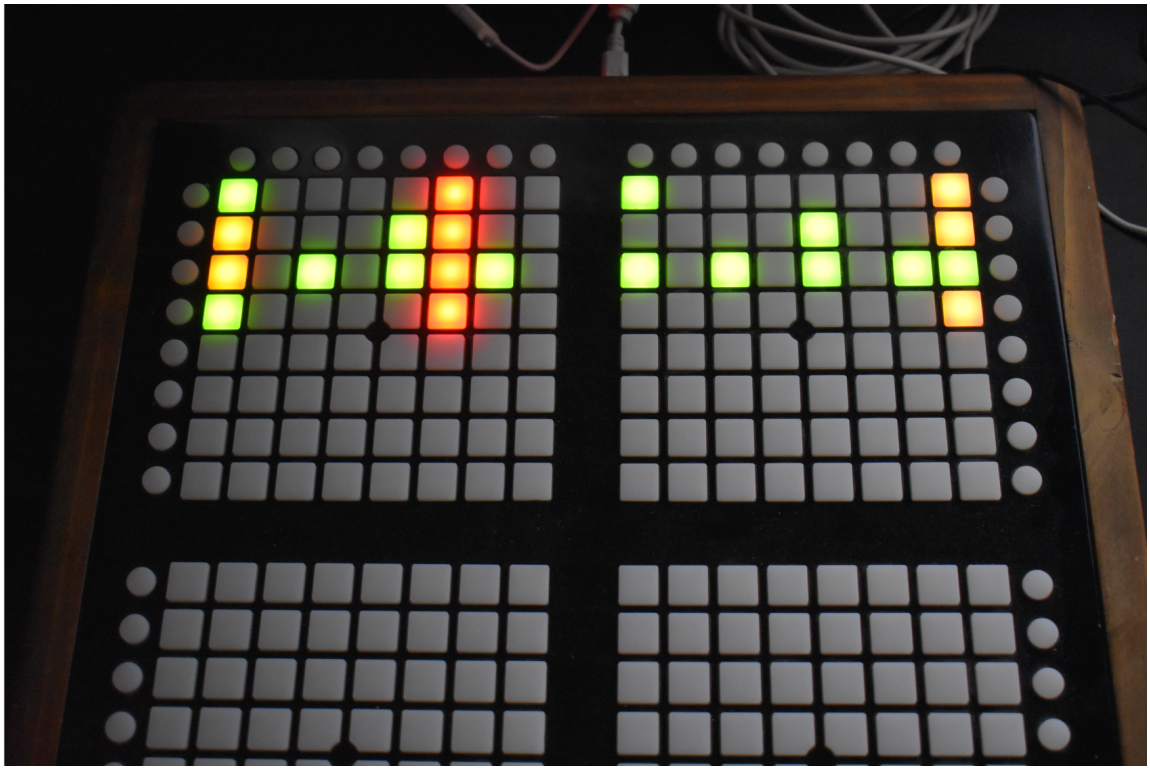

Fig. 6. Basic step sequencer, realisation of the code in figure 6

the concepts, and provides more details than is possible in paper[7]. Additionally, the range of possibilities are far greater than that demonstrated here as we focus on providing relatively simple instruments that promote demonstrability over complexity.

4.1 Polyrhythmic examples

To showcase polyrhythmic controllers (see figure 7) we need to create two (or more) step sequencers that are always in sync on the first step. In order to set up a 4 over 3 polyrhythm we would create two step sequencers one that has 16 steps and one that has 12 (any number of steps would work as long as the ratio between them is 4/3). Each step-sequencer requires a different clock, in this example the first has a tempo of 120 and the second 90 (25% slower). In any polyrhythmic container the ratio between steps (16/12) and clock speed (120/90) are proportional.

The software constructs these containers and computes relative clock speed automatically and a user simply needs to supply the number of steps for each of the two sequencers. In the example, the top of the grid has the 16 step sequencer and the bottom of the grid the 12 step sequencer, with each controller having 8 voices. LED feedback is used to show the start and end points of each sequencer. With little effort we could easily create a sequencer that is a 16/15/13/11 polyrhythm, which is considerable complex, although its musical merit remains something for the user to evaluate.

[7]https://www.youtube.com/playlist?list=PLES5ig6CvJKTYDKepwLWc37nniWoP7gwX

One downside of this representation is that (using the example shown at the start of this section), the 3/4 rhythm is shown to be shorter that the 4/4 rhythm, when in reality these are the same length. This is of course due to the limitations of needing a button for every time step. A continuous interface such as a touchscreen would allow this easily, but is not the focus of the project.

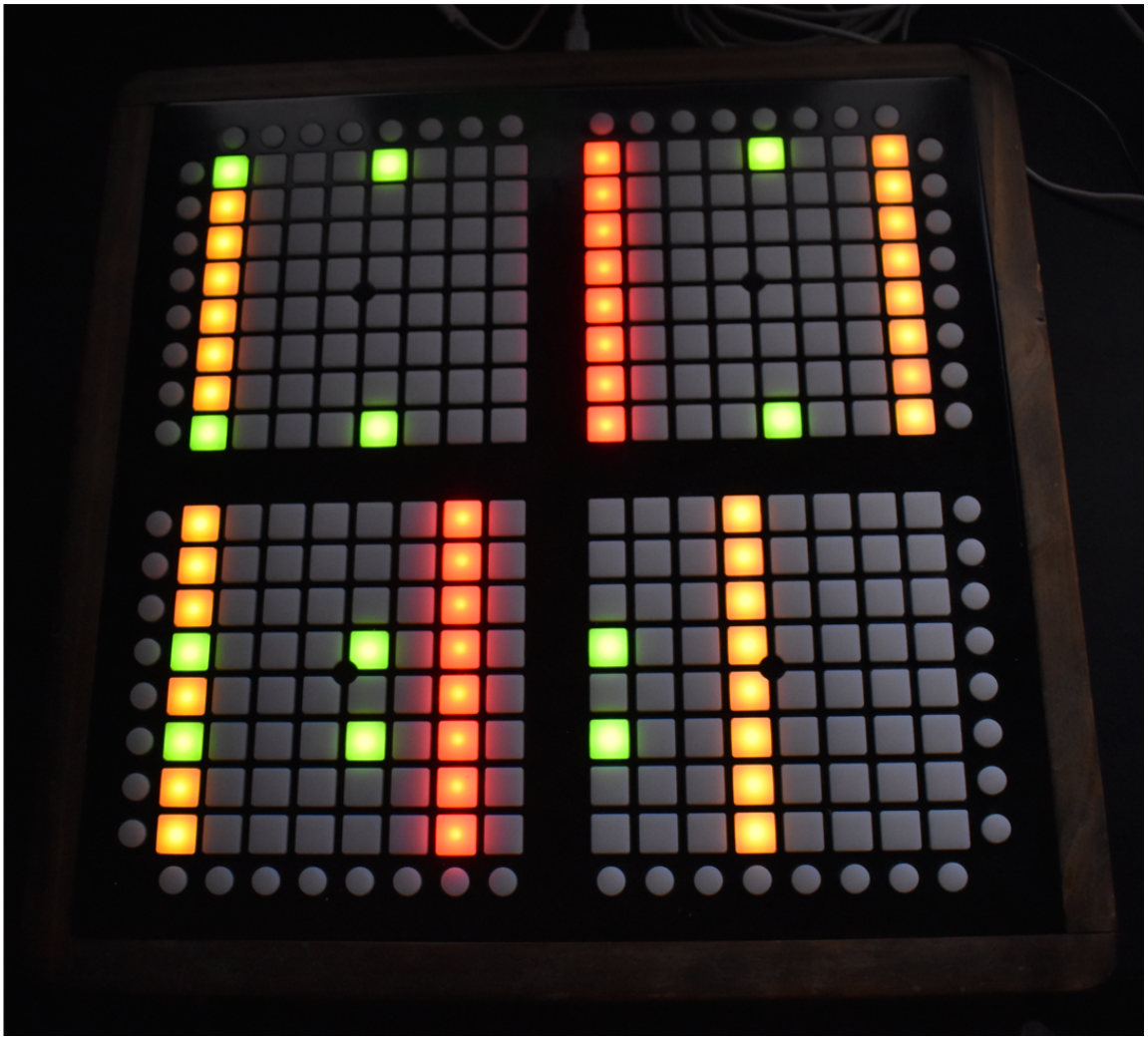

Fig. 7. 16/12 polyrhythmic sequencer, class: AMSequencer2WayPolyrhythmicContainer

4.2 Polymetric examples

Polymeterical examples are easier to configure as the sequencers share the same clock, but have a different numbers of steps between sequencers. In the example (figure 8) we have a 16 step sequencer played against a 10 step sequencer,

giving us a 4/4 over 5/8 polymeter. As these are shown together visually, the user can observe the sequences going in and out of sync.

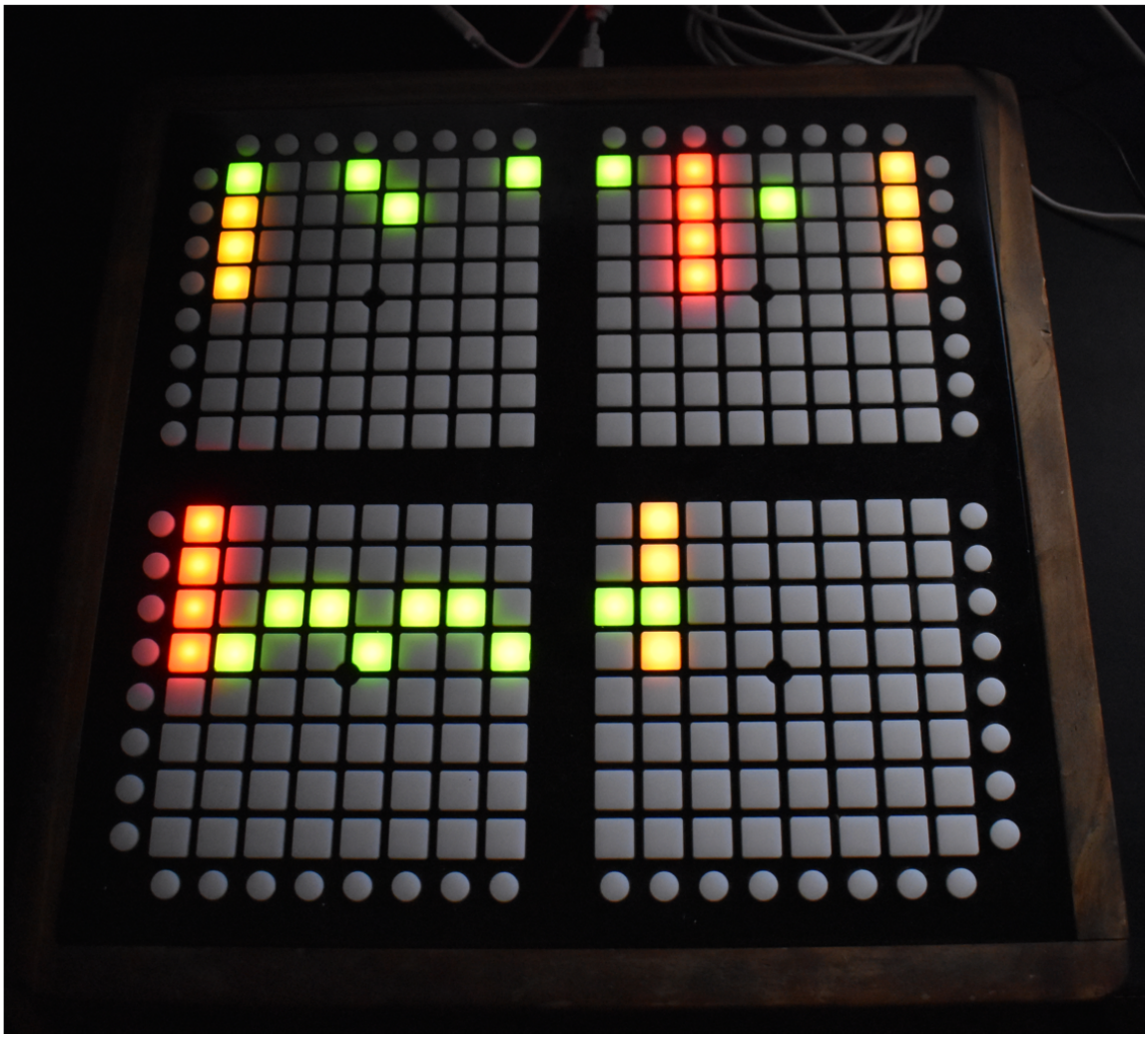

Fig. 8. 4/4 against 5/8 polymetric sequencer, class: AMSequencerPolymetricContainer

4.3 Polytempi examples

For the polytempo example we created 8 separate sequencers with a single voice (with each row outputting a separate note) and all 15 steps in length (see figure 9). Each sequencer was given its own clock with a tempo 1 bpm faster then the previous one, starting at 120. The lower rows appear to speed up as the pattern progresses, thus illustrating the concept visually. In this example it would take 4 minutes for the piece to repeat.

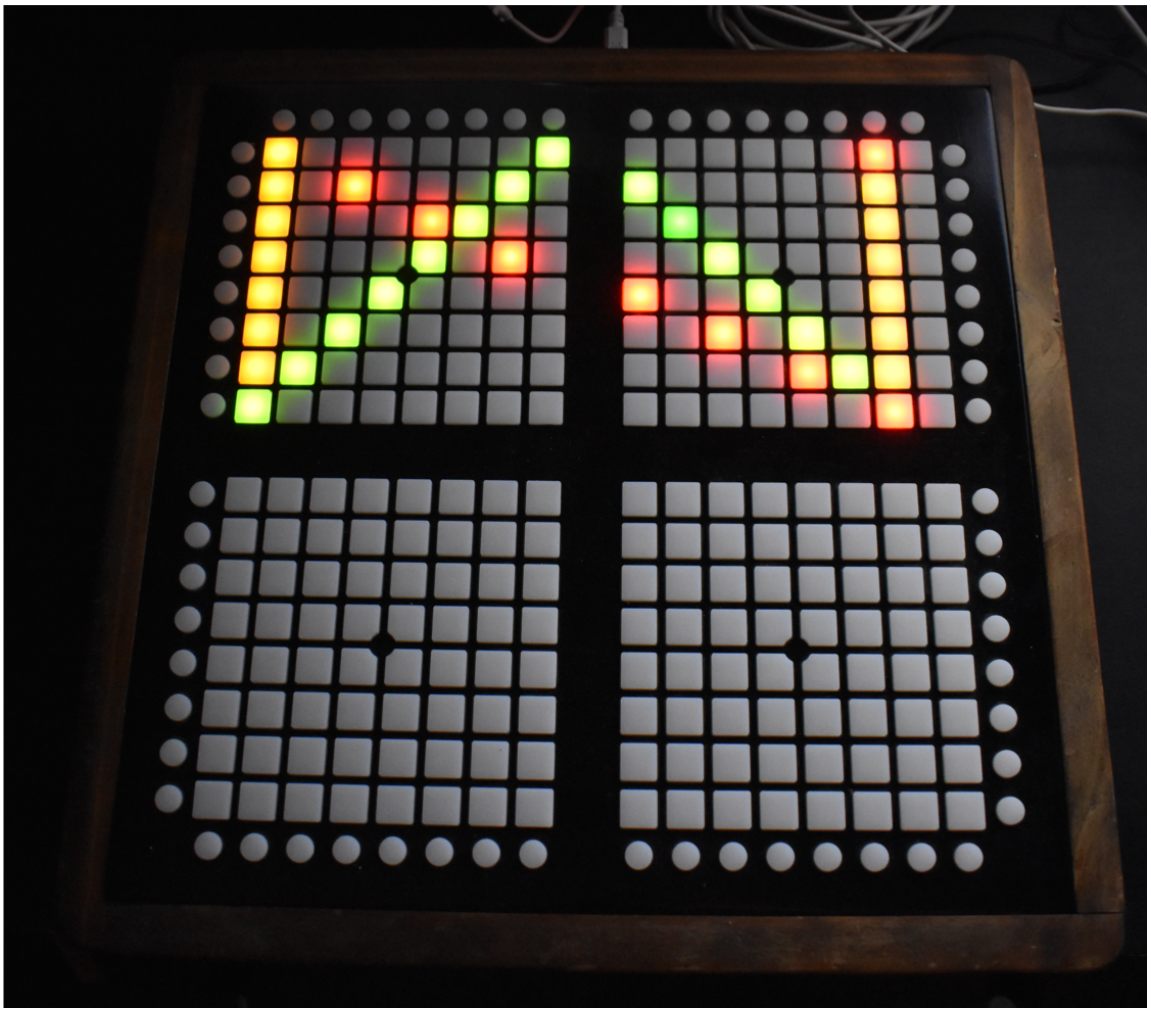

Fig. 9. 8 step sequencers running in a polytempi configuration, class: AMSequencerPolytempoContainer

5 CONCLUSION

This research sets out the initial objectives of the UGS framework and discusses its architecture. We encourage the development of the system to support more grid controllers and the creation of novel instruments. The project is provided as open source software. The framework provided us with a platform from which to develop and easily create step sequencers for exploring polyrhythms, polymeters and polytempos. With more complex examples of rhythms seemingly stochastic music was easily created, however the focus of this research was not on the musical quality of the output. Mathematically the music can be modelled and described, although it is unclear audibly whether a listener can perceive these structures, and would warrant further study and research.

Future work should therefore be primarily focused on evaluating the research with empirical user studies. Many activities are permissible, however emphasis should be placed on gauging a users understanding of complex time and

rhythmic relationships, and the ease at which such interfaces can be assembled. Other studies could include more open ended exercises focusing on what a user chooses to do with these tools and to what extent they felt artistically liberated by using them.

REFERENCES

[1] Willi Apel. 2003. *The Harvard dictionary of music* (4th ed.). Harvard University Press, Cambridge, Massachusetts.

[2] Christopher Dobrian. 2012. Techniques for Polytemporal Composition. In *Proceedings of Korean Electro-Acoustic Music Society's 2012 Annual Conference (KEAMSAC2012)*. Korean Electro-Acoustic Music Society, Seoul, Korea, 1–8.

[3] Alan Dorin. 2002. LiquiPrism: Generating polyrhythms with cellular automata. In *Proceedings of the 2002 International Conference on Auditory Display*. Georgia Institute of Technology, Advanced Telecommunications Research Institute, Kyoto, Japan, 447 – 451.

[4] Jared Dunne. 2007. Monome 40h Multi-Purpose Hardware Controller.

[5] Martim Galvao. 2014. *Metric Interplay: A Case Study In Polymeter, Polyrhythm, And Polytempo.* Ph.D. Dissertation. UC Irvine.

[6] Dave Hodder. 2018. *Launchpad Pro.* Technical Report. Focusrite. https://github.com/dvhdr/launchpad-pro

[7] Viljo Malmberg et al. 2010. *Iris: A circular polyrhythmic music sequencer.* Master's thesis. Aalto University.

[8] Monome. 2020. *Grid.* Technical Report. Monome. https://monome.org/docs/grid/

[9] Chris Nash. 2014. Manhattan: End-user programming for music. In *Proceedings of the International Conference on New Interfaces for Musical Expression*. NIME, Goldsmiths, University of London, 221–226.

[10] Nathan Renney and Benedict R Gaster. 2019. Digital Expression and Representation of Rhythm. In *Proceedings of the 14th International Audio Mostly Conference: A Journey in Sound*. Association for Computing Machinery (ACM), Nottingham United Kingdom, 9–16.

[11] Richard Vogl and Peter Knees. 2017. An intelligent drum machine for electronic dance music production and performance.. In *17th International Conference on New Interfaces for Musical Expression*. NIME, Copenhagen, Denmark, 251–256.

[12] Scott Wilson, David Cottle, and Nick Collins. 2011. *The SuperCollider Book.* The MIT Press, Cambridge, Massachusetts.