An App Design for Complex Rhythm Sequencing

Regis Verdin
Georgia Tech Center for Music Technology
840 McMillan St.
Atlanta, GA 30332
1 (415) 305-9212
regisverdin@gmail.com

ABSTRACT

This project involves the design and implementation of a music sequencer app for iOS. The app seeks to allow a novel approach to composing rhythms. The possibilities and limitations of music hardware and software play a large role in shaping and guiding the creative possibilities when using that technology. It was observed that electronic music sequencers tend to fall into two categories: quantized to a regular pulse, or completely un-quantized. The app presented here explores an approach somewhere in the middle: the user can define their own irregular quantization patterns, and add audio clips to the available steps in these patterns. This paper discusses the design considerations, the details of the implementation, and an evaluation of the usefulness of the app's novel features.

1. INTRODUCTION

1.1 Rhythmic Complexity

Rhythm is a highly expressive aspect of music, and the author believes that more tools could be made to facilitate new kinds of rhythmic expression and meaning. These tools could introduce the subtleties of rhythm found in human acoustic performance into music created with digital means, while also opening up possibilities that would be difficult or impossible for humans to perform (moving beyond the computer simply imitating human performance). This project aims to develop a tool like this, with its basis in music sequencers.

The app attempts to expand the user's expressive and creative possibilities by making more complexity and subtlety available in the way rhythms are crafted. Rhythmic complexity has been deeply explored in many areas of music, including the realm of iOS music apps which this project inhabits. This app, however, seeks to allow the user to explore rhythm in a more continuous time domain rather than a discretized one.

1.2 Music Sequencers

Automated devices for sequencing musical events have existed since as early as the 9th century. Since the industrial revolution, mechanization and automation have played ever increasing roles in our lives, and this has extended far into the realm of music. Precision, regularity, and speed are what computers excel at, and it is no wonder that the software for creating music with computers often reflect this bias. Step sequencers in particular reflect these characteristics, with their quantized step sizes and agility at playing repetitive patterns.

There has been some effort in the commercial realm to create electronic music tools that sound less rhythmically rigid, and more human. This is often achieved by shifting the step onset times to create swing. In some other cases (as with Finale's Human Playback) the attempt at eliminating rigidity goes beyond solely rhythm, and applies to dynamics and phrasing as well. The app designed here explores the extension of these rhythmic ideas from step sequencers, by allowing novel types of swing and other continuous-time rhythmic alterations to be used as an integral part of the music creation process.

1.3 Research Goals

The primary goal in this project was to design and prototype a user interface for a new type of rhythm sequencer that could be used to generate complex rhythms, with a focus on rhythms in a more continuous time domain than discrete. The app draws influence from step sequencers in that audio clips must be placed on a quantized grid. However, the quantized grid can be defined and modified in a number of ways not normally possible in step sequencers. While rhythmic complexity can certainly be achieved using discrete methods and even subdivisions, the goal in this project was to break away from those models of rhythm.

Target users for this app are musicians with an interest in composing using novel music-theoretical techniques, likely working within the context of experimental music. To allow the user to focus on composition more than implementation, the goal was to design an interface that provided a simplified workflow. The primary use-case for the app is composition rather than live performance or improvisation.

Certain design requirements were considered integral to the project, and were used as metrics of the project's success. First, the system should be capable of reproducing un-quantized subtleties of human rhythmic performance. To achieve this, the system takes the notion of temperament from tuning as an analogy. Finally, the system should be able to not only achieve these human-sounding rhythms, but also extend them to generate novel rhythms.

2. RELATED WORK

2.1 Related Music

The inspiration for these ideas came largely from hearing existing musical examples, and trying to understand how the rhythms in them could be generated with a computer and extended beyond their human form. In particular, the focus here was on rhythms that could not be well described as rational, or as not having a perceptible unit of subdivision (if any at all).

Swing in jazz is an example of a rhythmic performance practice that fits the above description. When a piece of music is played with swing, pulses of the same *notated* duration will have different *actual* durations when performed. A typical (if somewhat reductive) view of this is that every 2nd eight-note in the measure is shifted towards the 3rd note in an eight note triplet. The degree to which these notes are shifted indicates how "heavy" the swing is.

A more complex example of a swing-like rhythmic feature can be found in the Tigrigna music of Ethiopia and Eritrea. A recording of Tsehaytu Beraki's "Bazay" **[1]** reveals this unique rhythm. A subdivision into five pulses can be heard. The pulses are not evenly spaced: they are roughly grouped in two parts, the first three as triplet-8th-notes, and the last two as 8th-notes. Altogether, however, the pulses are also "swung" towards five quintuplet- 8th-notes. The emphasis varies depending on the section of the song and the instrument. For example, the bass plays closer to the triplet values, while the guitar plays somewhere in the middle. Later in the song, the entire ensemble shifts closer to a straight 5 feel. The unit of subdivision seems to change throughout the measure, and throughout the piece of music. Its flexibility serves an expressive and structural role.

Interestingly, modern popular Tigrigna music does not reveal much or any variation in the pulse length, which is likely the result of it being produced with step sequencers that cannot achieve that effect. This highlights a concern that the ease of making quantized music might be encouraging musicians to sacrifice a certain level of depth found in human performance practice.

Conlon Nancarrow's music is also relevant here, as a number of his Studies for Player Piano have irrational rhythms in a literal sense: they are generated from irrational numbers. For example, his Study 33 Canon 2 **[2]** features a sequence of 2-unit pulses superimposed on a sequence of √2-unit pulses. The relationship between these two lines is irrational, so there is no possibility of subdivision into a regular step size that accommodates both sets of pulses. Nancarrow's music reflects a desire to extend rhythmic complexity far beyond normal human capabilities.

2.2 Related Music Theory: Tuning

The construction of rhythm can be informed by analogy to tuning systems. While there are clear differences in the perception of pitch and rhythm (the former being perceived on a roughly logarithmic scale, the latter linear), the analogy is still worth investigating.

To approach creating human-like rhythms, some approximations in timing are desirable. Temperament in tuning features a kind of approximation that was taken as a starting point here. In order to address the goal of approaching human rhythms, the notion of tempering commas was applied to rhythmic beats (not the acoustic meaning of beat), in order to generate new beat timings.

Some background on temperaments: tuning systems are generally classified as either just-intonation or as temperaments. Just-tunings have rational intervals, while temperaments trade off having rational intervals for some other feature, such as intervallic consistency in transposition. In just-tunings, there is a notion of "harmonic limit": a limit is the highest prime number that is present as a factor of (or is used to generate) the ratios of intervals in the tuning system. **[3]**

Figure 1 shows two sets of just intervals (top and bottom rows) being tempered to create a new set of intervals (center row). The top row is a 5-limit system, the bottom row 2-limit. The comma being "tempered out" is the small interval between 3 just-major thirds ((5/4) ^3) stacked and the octave (2/1), also known as the diesis.

The tempered set in the center gives access to two interpretations of intervals. For example, a musician now has a note C that roughly stands for both "the pitch a just-major third up from G#" and for "the octave above the lower C". These two interpretations are not reconcilable without tempering. Tempered intervals gives access to several approximated prime generators (like 5 and 2 in figure 1), which can increase the harmonic complexity by allowing ambiguity through harmonic double-entendres.

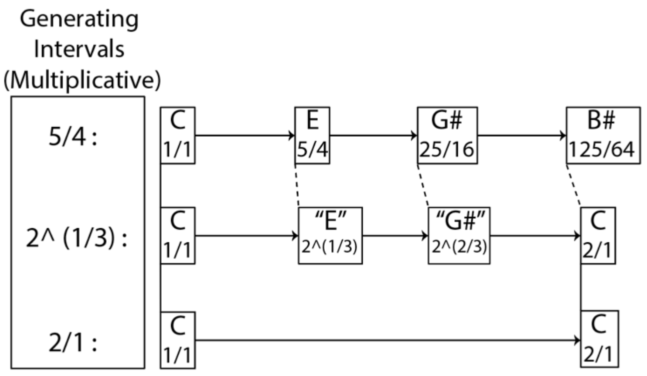

Figure 1. Tempered Tunings

This idea can be roughly carried over into the rhythmic domain: by slightly lengthening or shortening a unit of the music's pulse, a different generating subdivision can be suggested. Clearly this can be done without any shifting, as in hemiola (using a change in accent pattern), but this project focuses on cases where the different suggested subdivisions are coprime (as in 5 and 7). Figure 2 shows this rhythmic temperament process being applied to subdivisions of 5 and 7, with values being selected exactly in between their corresponding beats.

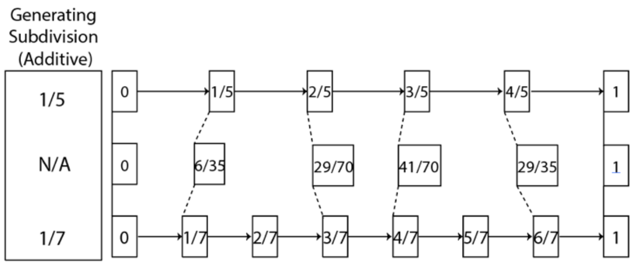

Figure 2. Tempered Rhythms

The choice of beats to map onto each other is somewhat arbitrary, although perhaps they could be chosen based on concepts like "evenness": the beats chosen for mapping from the top and bottom set could form a Euclidean rhythm **[4]** within their respective subdivisions. One might also consider that the center rhythm is not technically irrational, with a subdivision into 70 parts. However, it effectively is because this subdivision is too fast for a human musician to perform or hear. Note that the tempered rhythm does not necessarily need to be exactly in the middle of the pair of beats from both patterns: it can be interpolated gradually. Also note where the analogy somewhat breaks: in the rhythm domain when we "temper" we go from even divisions to uneven divisions, while the opposite is true in the pitch domain.

2.3 Related Music Technology

Many step sequencers allow the user to add swing, even on the simplest of drum machines. A knob can usually be turned to adjust the amount of swing. A more advanced sequencer like *Ableton Live* allows the user to select from a number of preset "groove" patterns, where the length of each beat varies depending on the selected style of groove.

There are also many existing music technology products that accommodate discrete forms of rhythmic complexity. These tend to be geared towards generating polyrhythmic or polymetric music. In the iOS music world, one of the most popular apps with the potential to create complex rhythms is *Patterning* **[5]**. It is a circular grid-based step sequencer that allows user-defined polyrhythms and polymeters, and a number of other interesting rhythmic effects such as rotating the starting time on each measure. Another app named *Sector* **[6]** uses a transition matrix as a model for ordering audio clips, which in itself can create interesting rhythmic arrangements. For instruments that allow generative approaches to music creation like *Rhythm Necklace* **[7]**, Euclidean algorithms are commonly found.

3. METHOD

3.1 Abstraction of Swing

An abstraction of swing was taken as the starting point for the design. A regular jazz swing can be seen in figure 3. This can be made more musically interesting if one considers the process of transforming the swing; each of these iterations in the two note-pattern interpolation have a unique sound. The transition itself could become a musical feature, occurring over any number of iterations.

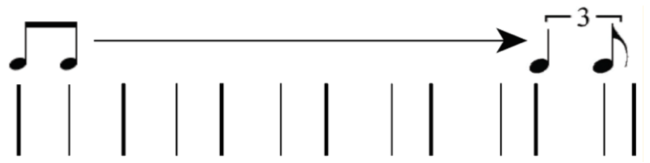

Figure 3. Swing

When treating swing as an interpolation between n-note patterns, rather than just 2-note patterns, we can find a way to model rhythms like those found in Tigrigna music. For example, figure 4 shows an interpolation between two different 4-note patterns. There is also the possibility of transitioning between patterns that have different numbers of beats. This could involve mapping multiple beats from one pattern onto a single beat on the other.

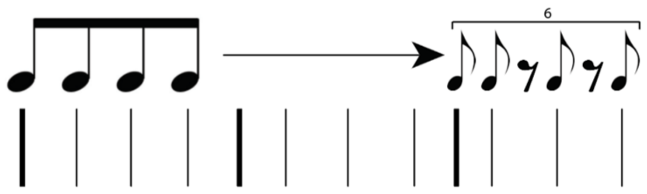

Figure 4. Swing Abstraction

3.2 Design Description and Considerations

The design of the app consists of roughly 4 screen sections: a timeline view, transport control, tool panel, and a tabbed audio clip selection panel. The app was limited to iPad-only because of space constraints on smaller devices.

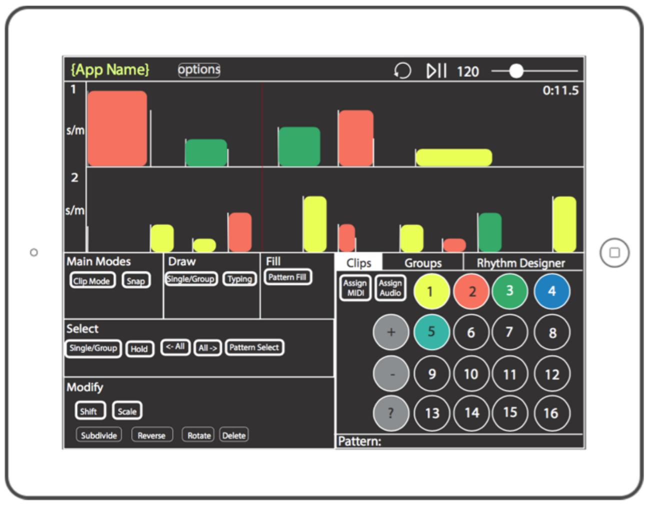

Figure 5. App Design

The design was conceived largely from deciding which musical possibilities should be allowed, then creating walkthroughs to see what features should be added to accommodate the musical requirements.

The timeline consists of horizontal tracks containing two elements: grid markers and clips. Grid markers define locations in the timeline where events can be triggered. They are similar to the quantization grid in many DAWs and sequencers. However, their locations are customizable and must be set by the user. In the appropriate mode, the user can tap a location in the timeline to add a grid marker. Guides should be shown to help the user align elements between tracks.

The height of each grid marker is determined by the user's touch location when adding it, and indicates the playback amplitude for that location. Linking amplitude to the rhythmic grid, rather than specific clips in the grid, was done to facilitate making sequences of accent patterns that could be copied and repeated. Dynamic patterns were felt to be a significant component of rhythm, and so the association between the two is somewhat fixed in this design (although there is certainly room for manual adjustment of amplitudes). Audio or MIDI clips can then be placed on grid markers. This associates the clip with a time and amplitude in the timeline.

The transport control design is fairly standard for audio software, and contains the following: a *play/pause* toggle button, a *back-to-*

beginning button, a tempo slider, and a volume slider. The definition of tempo here is somewhat abstract as there is no fixed beat-length; instead, it is simply displayed as a unit-less scaling factor for all the time intervals. This makes synchronization with other music software somewhat of a challenge, but for now the design treats this app as the master clock.

On the bottom left is the tool panel, which is divided into several sections: Main-Mode Selectors, Draw Tools, Fill Tools, Selection Tools, and Modification Tools. *Clip Mode* and *Snap* are state changing toggles that affect most of the other buttons. *Draw single/group* lets the user tap new grid markers into the timeline. If *Clip Mode* is toggled, then this allows the user to choose a clip from the clips panel and add it to an existing grid marker in the timeline. If *snap* is not selected, then the user can add a clip and a grid marker at the same time to any location. *Typing draw* allows the user to type in the clips panel and have that clip be added to the next available grid marker.

Pattern fill lets the user type a code (using the numbers in the clip panel) that automatically adds a sequence to the timeline, sequentially placing clips on grid markers. The code displays in the *Pattern* area, and specifies a rhythm with a sequence of pairs: clip number, followed by number of repetitions. The – *(minus)* key can be used in place of clip number to specify rests, while + *(plus)* is used to sustain the previous clip.

The selection tools do as they say: select either grid markers or clips in the timeline (depending on main modes). *Hold* adds to selection, working somewhat like a shift key on a computer keyboard but also inverting the selection. The *All* buttons select all elements left or right of the tapped point. Pattern select uses a similar code as *Pattern Fill* to specify which timeline elements should be selected.

Assuming the user has made a selection, they can now use the tools in the *Modify* section. *Shift* lets the user drag selected clips left or right. If clip mode and snapping are enabled, then this moves the assigned clips to the previous or following grid markers without shifting the grid markers themselves. *Scale* is used to resize the selection by expanding or contracting it. If an outer grid marker is selected as a handle, the entire selection is resized by dragging. If a grid marker in the middle of a selection is used as a handle for scaling, then the outer selected elements are fixed and the inner elements are moved proportionately with the dragged handle.

The remaining buttons are not toggle buttons but one-time modification buttons. *Subdivide* adds n-steps in between each selected element, where n is determined by the selected number in the clips panel. *Rotate* shifts all selected elements forward and places the last one at the front of the selection. *Reverse* and *delete* are self explanatory.

In bottom right corner we have 3 tabs: *Clips*, *Groups*, and *Rhythm Designer*. The *Clips* tab allows the user to assign audio clips or MIDI note numbers to any of 16 clip spaces. + and – are used as part of the pattern code to indicate "hold previous note" or "rest", respectively. *?* picks a clip number at random (and picks a new random value every time a clip is placed).

The *Groups* tab in Figure 6 works much like the *Clips* panel, except that it holds groups of grid markers or clip sequences. This was meant to encourage reuse of material, as the user can add a selection to this bank at any time. This takes the place of a copy/paste button.

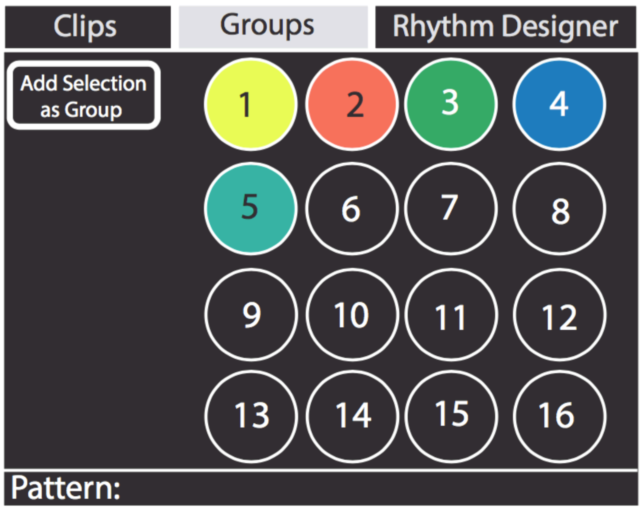

Figure 6. Groups

The *Rhythm Designer* shown in figure 7 allows the user to create rhythmic temperaments. The user first should create the top grid markers by selecting a subdivision (like 7 or 5), and choosing which beats should be part of the final template. The user repeats this process for the bottom template, then chooses which beats should be mapped to each other by dragging arrows. Finally, the green slider handle can be dragged to choose the amount of interpolation between the two subdivisions, resulting in rhythmic timings corresponding to the blue dots along the center row.

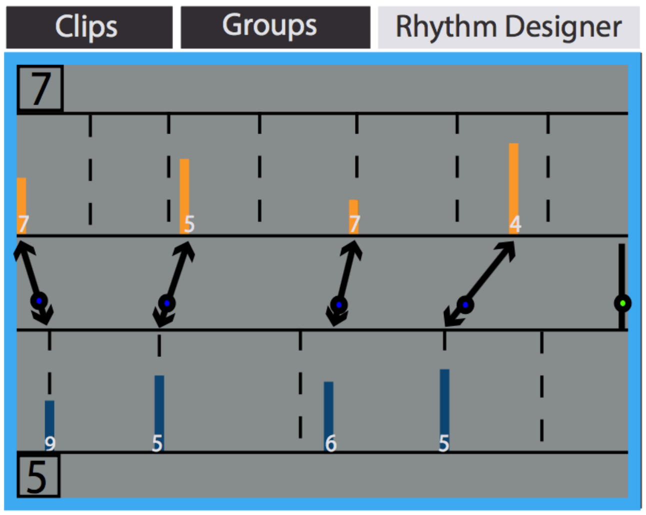

Figure 7. Rhythm Designer

These graphics were intended as low-fidelity wireframes of the app for prototyping purposes, so many important aesthetic decisions were glossed over. One significant consideration here, however, was the background color: a dark theme was chosen to accommodate use in dark rooms, which is a common music performance setting.

3.3 Implementation Description

A basic prototype of the *Rhythm Designer* panel was implemented in Max/MSP to decide whether the rhythms it could produce might be musically interesting or worthwhile. Following this, development in iOS became the main task. Ionic, a framework for developing iOS apps using web technology, was initially considered but abandoned due to lack of reliable audio performance and timing. A prototype was implemented using Objective-C. The SpriteKit framework was used to handle animation of the timeline, and The Amazing Audio Engine 2 framework **[8]** was used to schedule and play audio. The program structure is outlined in figure 8, and uses the Model-View-Controller design pattern as required by Apple.

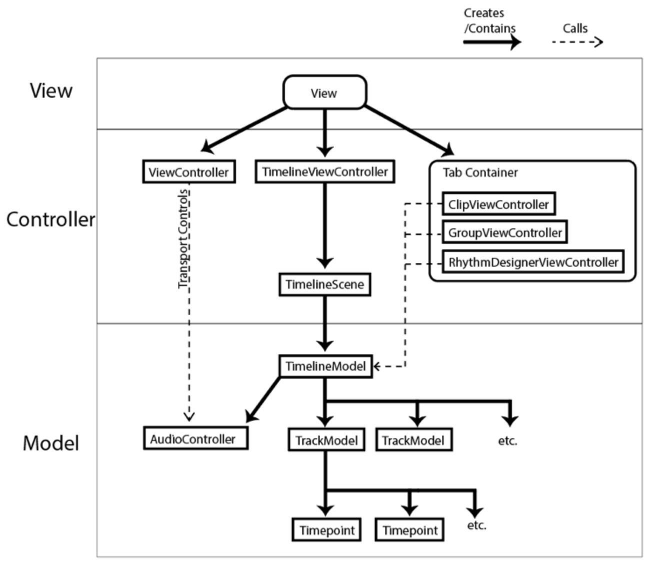

Figure 8. Program Structure

The transport controls have *play/stop* (no pause yet) and *loop* functions. An *import audio* button is used to test loading of external audio files using Kymatica's *AudioShare* app **[9]**. This will eventually be the main way to manage audio files for this app, but for now the audio files are loaded into the app's memory with no way for the user to access them.

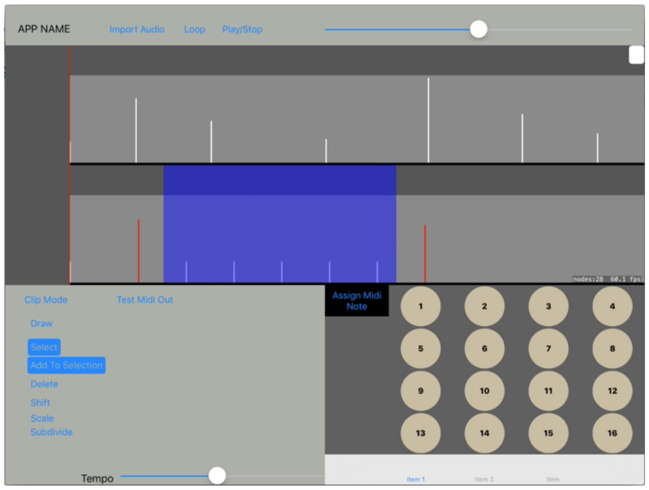

Figure 9. Prototype Implementation

The timeline view can be seen with two tracks available, the default number. The user can add grid markers and clips as described before. Selections can be made, with *Add to Selection* taking the place of the former term *Hold*. The modification tools *Shift, Scale, Subdivide,* and *Delete* all have basically functionality implemented. Figure 10 shows a selection of red grid markers, initially evenly spaced with the *Subdivide* tool, being *Scaled* with the blue grid marker as an anchor. The red vertical line is the playhead.

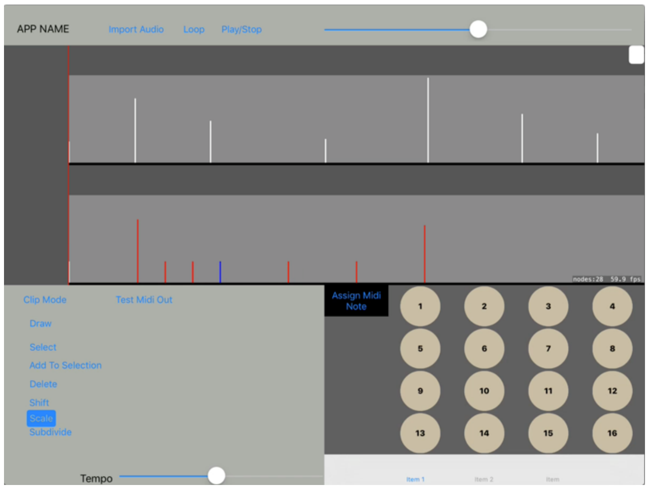

Figure 10. Subdividing and Scaling

Assign Midi Note can be toggled, allowing the user to tap a clip number and then select a MIDI note number for that clip from a UIPicker (see Figure 10). The groups and rhythm designer tabs have not yet been implemented, along with several other features such as the *fill* and *pattern*-related tools.

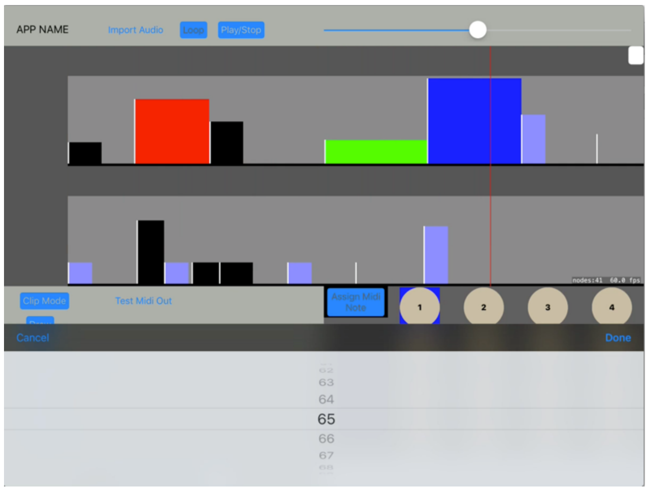

Figure 11. Assign MIDI Note Number to Clip

4. DISCUSSION

4.1 Evaluation of Programming

Many challenges were met while attempting to build this prototype, mostly resulting from the author's inexperience with iOS and Objective-C/C++/C programming. Once the basic layout was implemented, the backend of the TimelineScene took the most time. The app's toggle states are maintained successfully, with buttons being deselected when their state no longer applies to a new button selection.

One issue that repeatedly came up while building the app is the difficulty of accessing certain methods and properties of objects not owned by the calling object. The program structure reflects an initial lack of understanding that Objective C objects do not have "class variables" (as opposed to instance variables). These can, however, be implemented using C++ static variables, with several caveats. The object creation structure, which is a chain from TimelineScene down to the individual TimePoints, also contributes to these access difficulties. It resulted in many redundant methods that call down the chain. This results in several points where MVC is not applied consistently.

The implementation in its current state contains a number of bugs which should not be problematic to fix. For example, when dragging elements in the timeline with clips in it, the clips resize incorrectly when grid markers overlap with them. A more significant bug involves the audio: there is a load click at the beginning and end of every audio buffer loaded into the timeline, which is disruptive and made it difficult to evaluate the musical usefulness of the app.

A major refactoring is also needed regarding the various data structures holding time-points. Aside from the array containing these points in the AudioController, the time-points are not generally inserted in sorted order. This creates algorithmic complexity issues that affect the performance of modify tools in the UI (especially when dragging), because sorting is often used to find and compare or modify relevant time-points.

4.2 Evaluation of Implemented Design Features

The implemented design features were evaluated using a small set of expert users of music technology from the Georgia Tech Center for Music Technology, including the author. Users were initially asked to explore the app as though they were learning it on their own, while their actions and workflow were observed. They were asked to describe their actions as they took them, and explain why they made certain choices.

A basic analytical framework for evaluating interactive music systems was applied here. The input, output, mappings, and workflow were considered, along with the initial project's goals of creating various types of rhythmic complexity. The degree to which these features facilitated musical expression, subtlety, and enjoyment was questioned. A comparison with the complex drum machine app *Patterning* **[5]** was used to ground the discussion. Based on this, we consider the prototype's successes, failures, and limitations. These should guide future improvements.

The first element called into question was the app's requirement that everything be pre-composed, rather than improvised on the fly. Initially, there was no loop button but it became evident that this was one of the most important features (especially for testing purposes). Looping rhythms gives the musical creation process a sense of continuity without having a predetermined trajectory. This can be seen clearly in an app like *Patterning*, where the user can additively create a rhythm while continuously auditioning their material. Listening and performing are integrated into the composition process, which is more instantaneously rewarding.

The factor of immediacy is one that had not been regarded carefully enough here, but is very significant. Many iOS music apps require 2 or less buttons to be pressed before music is heard, which is the case with *Patterning*: tap a space in the circular grid, then press the play button and your first rhythm is heard. The workflow acts as a tutorial as well, as it allows the user to gradually incrementally explore the functionality without ever getting stuck in an unmusical moment. While this project's app did not set out with live-composition or improvisation as major goals, it seems that it should have been, given the iOS music app context and portability of the device.

In this kind of an environment, breaking the user's expectation is immediately negative if there is no clear promise of future gain from the lack of immediacy. In other words, simplicity and looping make music apps more fun. The current workflow in this project's app adds to this lack of fun and immediacy, as it requires the user to tap a precise sequence of about 6 buttons all over the screen in order to achieve a simple sound. Some of this could be mitigated by adding more touch-drag based features for adding clips to the timeline, rather than tapping buttons.

Patterning also contains a feature which is sorely lacking from this app: the ability to create and rearrange higher level structure from individual sequences, using the *Song* tab. A user can tap to add their currently looping sequence to a timeline (which itself can be looped), then continue to evolve their current loop without worrying about overwriting it. While this need had certainly been considered in this project's app, it seems that currently the only real option here is to rearrange everything on a low level that extends for the length of an entire song. Adding the missing *Groups* tab might help with this, but more support for large scale repetition and variation will likely be needed.

In terms of the rhythms, there were some successes. The rhythms are certainly in a continuous time domain, which fulfills one major goal of the project. The rhythms tested sounded somewhat human in their irregular timing. However, this highlighted a significant factor in what makes a rhythm sound natural or human: amplitude envelope of the sound. As it stands, there is no envelope placed on the clips, which inherently sounds machine like in its lack of subtlety. As for the goal of extending the rhythms into the realm of novelty, the app succeeds immediately, although it is difficult to tell what is a useful rhythm because of the audio glitches. The note-off click is especially disruptive, as it doesn't come at a user-defined moment and so is not predictable or controllable.

While the rhythms produced by the app certainly are complex and in a continuous domain, the interface still feels too close to a DAW sequencer with little rhythmic guidance. The author attempted to create some of the human rhythms discussed earlier, but without the *Rhythm Designer* implemented it was somewhat tedious and less successful. The Max/MSP implementation of the *Rhythm Designer* however did seem to create useful and musical rhythms, especially as the interpolation slider was moved to transition from one rhythm to another. Judging the sound of a rhythm from its visual position, which is the basis of the UI, gave unexpected and often interesting results.

5. FUTURE WORK AND CONCLUSIONS

This project was an exploration of novel rhythmic sequencing strategies implemented as an iPad application. The evaluation of the prototyped features suggested many design changes and bug fixes, which will be worked on in the near future.

The bugs described in the implementation evaluation clearly need fixing. Workflow speed and ease needs to be addressed, and the ability to create higher level musical form using repetition should be given higher priority. It could be interesting to sequence higher level sections using similar methods to those used in sequencing the grid markers and clips, especially if modification tools could be applied at the high level as well.

There is a question of how to convey rhythmic meaning to the user in this kind of timeline, without having a fixed grid. Some system of annotations to the timeline should help with this, suggesting alignments and possible similarity to various subdivision “limits”. On a related note, the user needs more guidance in this open world of continuous rhythms. A possibility is to have preset grids as a default on the timeline, allowing the user to clear or alter them if so desired. A blank page is the best way to writer’s block, and this is unfortunately what the current UI begins with.

Whether the rhythms have the musical merit that justifies the complexity of generating them has yet to be seen, although it seems likely that simplifying the workflow will help with evaluating this. Beyond the basic usability issues, the best way to truly evaluate the musical effectiveness of the rhythmic complexities described here would be through composition and improvisation. A series of compositions should showcase features and inform future developments.

This app’s design, prototype, and evaluation explored the possibility of expanding the standard rhythmic palette of step sequencers. Time in the real world is often perceived as repetitive yet in reality, these perceived repetitions come with infinite subtleties and variations. Heartbeats have a constantly shifting tempo, seasons appear at irregular intervals, and years need to be tempered with leap days to accommodate their discretization. The hope is to see more of this richness reflected in the world of music.

6. REFERENCES

[1] Beraki, Tsehaytu 1970. Bazay. *Ethiopiques, Vol. 5* (1970-1975) (reissue). Amha Records.

[2] Nancarrow, Conlon 1968. Study No. 33. *Studies For Player Piano* (1949-1988).

[3] Various authors. The Xenharmonic Wiki. (Jan 2015) http://xenharmonic.wikispaces.com/

[4] Toussaint, Godfried 2005. The Euclidean Algorithm Generates Traditional Musical Rhythms. *McGill University, School of Computer Science*.

[5] Kamen, Ben 2015. Patterning (app). *Olympia Noise Co.* http://www.olympianoiseco.com/apps/patterning/

[6] Liljedahl, Jonatan 2014. Sector (app). *Kymatica*. http://kymatica.com/Software/Sector

[7] O’Reilly, M., Tarakaijian, S. 2015. Rhythm Necklace. http://rhythmnecklace.com/

[8] Tyson, M. 2016. The Amazing Audio Engine 2. http://theamazingaudioengine.com/

[9] Liljedahl, Jonatan 2014. AudioShare (app). *Kymatica*. http://kymatica.com/Software/AudioShare