Trey Bradley

Voice-Driven Drum Machine

Abstract

This algorithmic composition final project brought together computer music tools like, Java Music Specification Language (JMSL), Java Synthesizer API (JSYN), Syntona, and knowledge that was gained over the course of E85-2608 Algorithmic Composition and Computer Music using Java.[1] The end result was a functional user-generated rhythmic compositional tool. The remainder of this report further describes this project, motivations for pursuing the topic, the methodology and technical description of this Java-based tool, and an artistic analysis and report of the tool's performance and creative capacity.

Introduction

Algorithmic composition is an expanding field of computer music that looks to harness software and hardware power, speed, and computational abilities to expand music beyond conventional tools, norms, and sounds. This specific field uses algorithmically-generated values to govern the most minute parameters of a sound, performance, and/or composition. As a crossroads subject, it brings together studies from various fields like digital signal processing, music information retrieval, computer science, electronic music, and composition. Articulated by Edgard Varese and Chou Wen-Chung, these digital technologies and electronic mediums can help fight for the liberation of sound from convention and its "paralyzing tempered system" (Varese 1966). Scientific and creative-based fields and practices like computer music and algorithmic composition can help expand creativity in music and redefine possibilities and musical norms. This notion presents the main motivation for pursuing this course work and developing a computer music composition tool.

Rhythm, usually set by drums or percussive instruments, provide a foundation to build on when composing a song or performing instruments. After establishing the tempo, pace, and energy of the composition (all captured by percussion) musicians can venture into a world of sounds, to build on their piece. Attention to the rhythmic component of a song provides the second main motivation behind this project—to develop a tool that can allow the user to easily set the foundation for their composition or live performance by composing drum loops or sequences, on-the-spot.

The main goal behind this project was to build a user-driven beat maker or rhythmic composition-generator that could live-generate a drum sequence (or loop), given a user-input. The user-input was aimed to be generated by a voice. In this case, the user could beatbox, sing, or speak into the program to generate the rhythmic components of a computer-music instrument. Performance, however, was best achieved by user-input sounds with sharper attacks and decays, like the sound of a pen or knuckle striking a table. The following sections provide a detailed description of the algorithmic composition tools and java programming scripts used to build the tool and optimize usability.

Detailed Description

Using the program is easy and intuitive. The user can begin by running the *BeatDetectingMusicJob.java* script. Upon running this script, the user is presented with the

[1] This course was taught by Nicholas Didkovsky (nick@didkovsky.com), using various tools from Phil Burk and Mobileer Inc. at http://www.softsynth.com/

graphical user interface (GUI) illustrated below in figure 1. After defining a tempo and rhythmic pattern, the user can then start recording rhythm for the first instrument, which is defined currently as a hi hat sample, by clicking the button 'Start Record'.

Figure 1 User's graphical interface / point of interaction upon running the program

After clicking the 'Start Record' button, the GUI transforms into the one in figure 2 with the 'Stop Recording' button enabled, and 'Start Record' disabled. At this point the user should sing or tap into the computer's microphone to begin recording the rhythm for the first instrument. It is imperative that they have the key signature and tempo in mind since they should time the clicking of 'Stop Recording' with the first downbeat of the sequence. This part is important to maintain the program's ability to loop (repeat) the sequence on beat and on time. Upon clicking 'Stop Recording,' the GUI returns to its original state (in figure 1), where the user can repeat this process for subsequent additions of rhythms and instruments.

Figure 2 User's graphical interface / point of interaction upon clicking 'Start Record'

After recording rhythms for one or all of the instruments, the user can click the 'Play' button, disabling the 'Stop Play' button as seen in figure 2, in order to playback their composition. The last element on the GUI is the 'Erase' button, which allows the user to erase their entire composition and start recording from scratch. While the 'Play' and 'Erase' buttons are enabled upon launching the program, clicking them will not perform any action until beats have been recorded and assigned to instruments to play.

Hidden from the user are features, further described in the technical description, that govern the beat detector, quantization to remove slight human error, and the loading of additional sounds and samples.

Technical Description

This section provides a walk-through of how the different computer music, composition, and java tools come together to build the project. All of the coding was done in Java. Five scripts and three audio samples (all described below and provided with this submission) are needed to run the program.

a. Peak.java

This code is the foundation of the tool. Developed in the JSYN API, this Java class allows the program to read peaks from any input (in this case the computer's default microphone). In this script, the user can tune the sensitivity of the peak detector by altering the 0.001 value on line 32 in `mPeakFollower.halfLife.set(0.001);` This half-life value determines the speed at which the peak detector approaches zero, after detecting a single peak in a sequence. This program returns time stamps of the detected peaks, which are used to build the beat further down the line.

b. BeatDetectingMusicJob.java

This program code makes beats using the time stamp values generated from peak detector. It is in two states of either recording beats or not, depending on communications from the action listener in the Jframe, from the 'PeakFollowerGui.java' script. Two important lines are key to highlight and can be manipulated by the user to adjust how peaks are defined and how beats are detected. On line 12, `final double THRESHOLD = 0.02;` can be used to adjust the beat maker's sensitivity to peaks amongst noise in an environment. Next, on line 46, the `delta` and `MAX_BEATS_PER_SECOND` values can be adjusted to manipulate the frequency at which peaks can be detected in sequence. The `POLLING_RATE` value on line 10 can also fine-tune this parameter. Finally, this program outputs a music shape with 1 dimension for duration, to be used when creating music shapes with assigned instruments in the 'PeakFollowerGui.java' script.

c. DrumSampleIns.java

This is a simple script that builds a sampler instrument by implementing the JYSN class `SamplePlayingInstrument`. This code is used to help sonify the user-composed rhythm and poses as chance for the user to interact and add a dimension of creativity in selecting samples. After establishing all of the imports and defining the directory to store all of the sample's files, the user can begin altering the program by stipulating which sounds they want and assigning values to those sounds, seen in the code excerpt below (and beginning in the code on line 13). These values are the pitches that are assigned to a specific sample, which can be called for in playback. The rational behind creating these samples as public variables, is for easy access in the main program, PeakFollowerGui.java, where the instruments can be called and added to music shapes. It is important to know that this class only reads .aif or .aiff audio files.

```
public static final int KICK = 35;
public static final int HIHAT = 80;
public static final int SNARE = 40;
```

d. PeakFollowerGui.java

This code is the core of the program—it brings together every individual element, script, and feature of the project and builds the graphical interface for the user to interact with. Upon running this script, all of the interface buttons are built and activated (line 62). Besides disabling the record button and stopping beat detection in the 'BeatDetectingMusicJob.java' script, the stop button also gets time stamp values from the 'BeatDetectingMusicJob.java' script (and converts them into durations), builds music shapes, attributes samples and instruments to the shapes, quantizes the durations in each element of each shape, and adds the shapes to a parallel collection for simultaneous playback of all of the newly-created music shapes.

e. Quantizer.java

This code is responsible for the functionality of the quantize feature, which removes some imperfections in the composition which tend to be erroneously generated by users who are slightly off-beat. Ultimately, this program requires two inputs: a music shape (with i elements and 0 dimension for duration) and the quantization value, as the core duration. The core duration stipulates the duration value of the first beat, which is the desired note to quantize all subsequent

beats (elements in the music shape) to. The quantizer rounds each duration in the music shape elements to the nearest integer multiple of the core duration. In the 'PeakFollowerGui.java' script, this is defined as quantizeValue, which can be further manipulated by dividing the core duration (elapsedTimeBetweenRecordAndFirstPeak) by a certain value, to achieve more precise quantitation values.

Artistic Analysis

At the point of writing this report, the current state of the project accomplishes the main goal of creating a rhythmic composition tool. Upon running the program, the user is given an easy-to-use interface, presented in figure 1. All of the bugs that are essential to the tool's functioning that were discovered throughout the building process have been solved, although some additional features and debugging can be addressed to add better usability and creative control. In this version, the user can compose beats of infinite length with four sample voices, including hi hat, kick drum, snare drum, and a tonal component (consisting of a punchy square wave instrument). Although the instruments are pre-set, the user can substitute other samples and/or instruments, with relative ease. After recording beats for the individual samples/instruments, the user can playback their composition, stop and resume play, and erase instruments. Ultimately, this current iteration allows the user to create a basic rhythmic foundation from which to introduce other acoustic and/or computer-music instruments to. With the features (and their imperfections) in their current state, the user has a fair amount of creative control and ability to achieve a range of sounds. The remainder of this section briefly describes additional features that are in the immediate pipeline for future iterations of the project.

a. Environmental Noise Detection

For one, the tool requires that the user tune certain parameters of the program to their specific environment so that peaks can be accurately detected to reproduce the desired beat. To address this user experience requirement of knowledge about one's sonic environment, artificial intelligence approaches can be implemented to automatically tune the device to environments with variable external noises and reverberations. Furthermore, this project can also borrow concepts from Music Information Retrieval (MIR) to select specific sound sources to use as inputs, in the midst of additional instrumental noise.

b. Score Writing and Fine-Tune Editing

Another feature that could add more creative control to the tool would be pair the device with JSYN's score writing capabilities. After recording a beat, this feature could transcribe the beat onto a playable score, which can be used to further manipulate and manually edit the beat, directly on the score's interface. Adding this visual component can aid in the rhythmic composition process.

c. Instrument Selection and Muting

In recording the beat, the user should be able to select specific percussive instruments to record and not be forced to record one after the other. Currently, in order to attain a desired instrument that may come after the first instrument (currently hi hat), the user is forced to record rhythms for every instrument until that one is given to the user as a choice. An additional GUI element can be added to allow the user to select which instrument that they want to compose for.

Further, the user could use this GUI element to be able to select certain instruments and mute others from playback.

In selecting specific instruments, the 'Erase' button should also be modified to not erase the entire composition, but only the instrument currently selected.

d. Effects Control

Any creative composition should allow the user to introduce any number of effects to the rhythm, like delay. Signal processing features would be an interesting way to increase usability and creative control for the user.

Conclusion

This project has a great future in allowing users to live-compose rhythms and play them back with ease. Granting them increased creative control through the implementation of the features discussed previously will ultimately allow for more experimentation and usability of the tool. In developing this project, knowledge about Java programming, computer music tools (like JMSL and JSYN), and various music technologies was essential. The JMSL and JSYN package and API allowed me to implement a number of features that are essential to the functioning. For example, the JSYN peak follower does all of the complex audio analysis to read incoming signals, detect peaks, and return them as beats in a parallel collection of music shapes, which are also Java-based composition tools. These parallel collections and music shapes allow for playback of user-selected instruments, where the user can specify a range of parameters like duration, pitch, and amplitude. Additionally, the sample playing instrument (another JSYN tool) allows the user to load samples of their choice, giving them freedom to achieve a certain sound. Ultimately this was an interdisciplinary project, bringing together skills and concepts from a variety of fields like computer science, computer music, and algorithmic composition.

Sources

Beginning Programming with Java for Dummies (4th Ed.) by B. Burd.

JSYN and JMSL documentation by Phil Burk , (available at http://www.softsynth.com)

The Liberation of Sound by E. Varese, C. E. Wen-Chung (1966). *Perspectives of New Music, 5(1).* pp. 11-19.