JavaBand

A Hybrid Multi-Instrumental Music Player with JMSL and Arduino

Jerry Xu

MPATE-GE-2608 Sec.001

Prof. Nicholas Didkovsky

12/20/2021

Abstract

JavaBand is a hybrid multi-instrumental music player created with JMSL (Java Music Specification Language) and Arduino aiming to experiment real-time computer music synthesis with controllable automation through Java. The project integrates some of the JSyn UnitVoice instruments which I have created from previous assignments into a band with a hardware interface. While performing, JavaBand plays a pre-programmed one-measure loop of 6 synthesizers whose parameters are controlled by a Teensyduino board of 2 joysticks through serial port. This paper will present a brief discussion on the product details, technical and artistic description as well as the direction for future works.

The link to the walk-through video and the link to the Google Drive folder.

Product Details

Prior to starting the application, the hardware controller must be plugged into the computer through USB serial port in order to get the full experience (Figure 1).

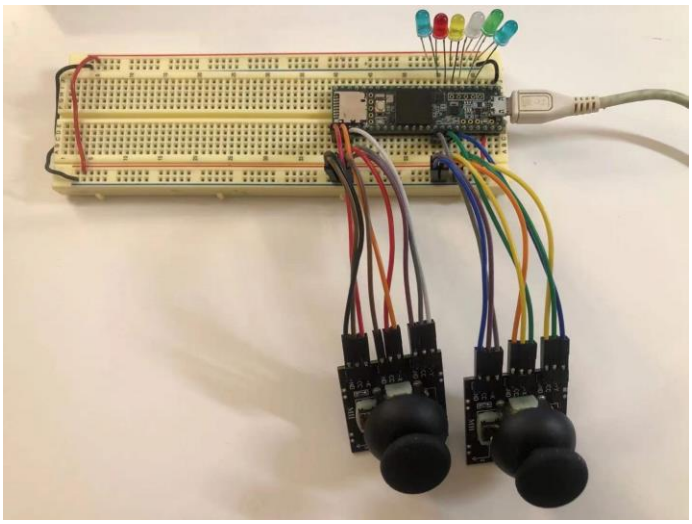

Figure 1. The hardware controller

By clicking the “JavaBand!” button on the GUI after running the program, JavaBand initiates the instruments and the serial port and starts the count-off with hi-hats (Figure 2). If it is run on a terminal, the user should be able to see the data from the controller constantly being printed on the console.

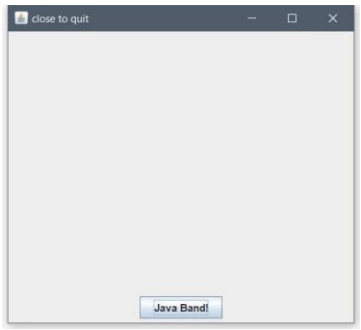

Figure 2. The Graphical User Interface

The LEDs on the controller indicates the current selected instrument. Starting from left is lead, pad, bass, hi-hats, snares, and kick respectively. By horizontal scrolling the left joystick, the user can switch to another instrument. To control the amplitude and panning of the current instrument, the user should click the left joystick to enter the EDIT mode as the LED turns bright. Under EDIT mode, horizontal scrolling of the left joystick will automate the panning of the current instrument while vertical scrolling will change the amplitude. Click the left joystick again to exit the EDIT mode as the changes are stored and the LED turns dark.

Clicking the right joystick will cue the lead instrument to play a random quarter-note phrase after 2 bars. Remain pressed to let the lead continue to play. While the current selected instrument is on lead, pad, or bass, scrolling the right joystick will automate certain parameters of the instrument. While lead is selected, horizontal scrolling automates the modulation

frequency; vertical scrolling automates the delay time and modulation amplitude. While pad is selected, horizontal scrolling automates the modulation frequency; vertical scrolling automates the modulation amplitude and the amount of filter. While bass is selected, horizontal scrolling automates the modulation frequency; vertical scrolling automates the modulation amplitude and low pass filter cutoff frequency. The automation will be applied in the next measure and will be stored if the user switch to another instrument.

To end the piece, simply click “Stop Playing” on the GUI. The band will be cued off after 2 measures. Otherwise, the user can close the application to shut down immediately.

Technical Description

The structural design of the JavaBand relies on multiple layers of encapsulation. The application, JavaBand with GUI, contains a MusicJob for serial port input which polls data from the USB serial port every 0.1 seconds with the support from the RXTX library and communicates to the main sound processing unit (Figure 3). The data would be sent to a MusicJob according to which instrument is currently selected by calling the setter method in it. Inside the sound processing unit, the MusicJob for the lead instrument is supposed to loop for each beat while the others loop for each measure of 4 beats, therefore a ParallelCollection is used wrap these MusicJobs for simpler data structure management. Moreover, a Mixer object is set up to sum the signal outputs from all MusicJobs within the sound processing unit and is set to receive changes of panning and amplitude from the serial data. Instead of using the super method “setTimeStrech(..)”, a convertor method “setTempo(..)” which calls inner “setTimeStrech(..)” methods of all MusicJobs is used for simpler musical composition. Upon clicking “JavaBand!”

on the GUI after starting the application, a “launch()” method is called to initiate all units, while “Stop Playing” would call the “finishAll()” method to wrap up all process.

Within each MusicJob unit for instruments, there is a MusicShape extension and a custom-made JSyn UnitVoice instrument which I have created in previous assignments using Syntona. Figure 4 illustrates the structural design of the MusicJob for the lead instrument. Upon each repetition, the MusicShape for the lead instrument will be re-written with preset pattern generation methods. The serial data MusicJob detects pressing of the right joystick and triggers a random number generator for the mode to play. There are a total of four modes to choose from: two straight eighth notes with upward melody, three quarter-note triplets going up and down, three sixteenth notes going up with a sixteenth-note rest, and four sixteenth notes going down. The pitches of each mode is determined by the “chordIndex” argument plus a random index of the “stepArray[]”. If a click of the right joystick is not detected, a quarter-note rest will be written in the lead MusicShape. After filling out the MusicShape, the data of each note will be sent to the JSyn UnitVoice Instrument to play using the “play(..)” method. In order to have a better sonic experience, lead MusicJob also contains a delay instrument which is a JSyn UnitSink instrument that takes a signal source as input. In this application, the delay instrument takes in the output of the lead instrument and outputs a delayed signal with delay time and feedback. In order to achieve the effect of real-time music synthesis with controllable automations, the MusicShape takes in current values of modulation frequency, modulation amplitude, and low pass filter cutoff frequency of the lead instrument as well as the delay time and feedback of the delay instrument when adding notes at the beginning of each repetition. As a result, the serial data MusicJob outside the class can access these values through the setters and apply the changes in the next repetition.

Similar structural designs are used for the pad, bass, hi-hats, snares, and drums, except that the percussive instruments do not have controllable parameters. For example, the MusicJobs for the snare and hats also utilize the idea of randomized pattern generation, which switch from the default pattern to the fill at the third bar of every four measures, creating an illusion of a four-bar loop. The bass part would also randomly switch between two melodic patterns to create liveliness throughout the performance. Moreover, by overwriting the "start(..)" method of each MusicJob, I was able to compose a two-bar hats count-off, making it more realistic.

The assembling and programming process of the Teensyduino board is simple. In order to get the scrolling data, the X and Y factors of the joystick modules must be set to analog inputs and properly scaled to 0.0 to 1.0. The clicks of the joysticks are set to digital inputs, of which the left joystick should be a switch-behaving button and the right should be a normal press-release button. Horizontal scrolling of the left joystick returns an integer as the index of the current selected instrument by a modulo calculator, which brightens up the corresponding LED and locks the number when the state of left joystick's button is on. The output serial data follows the format: "leftX leftY rightX rightY leftButton rightButton currentInstrument". The serial data operator in JavaBand parses the input data and prints it to the console using a "dump()" method for real-time monitoring purpose.

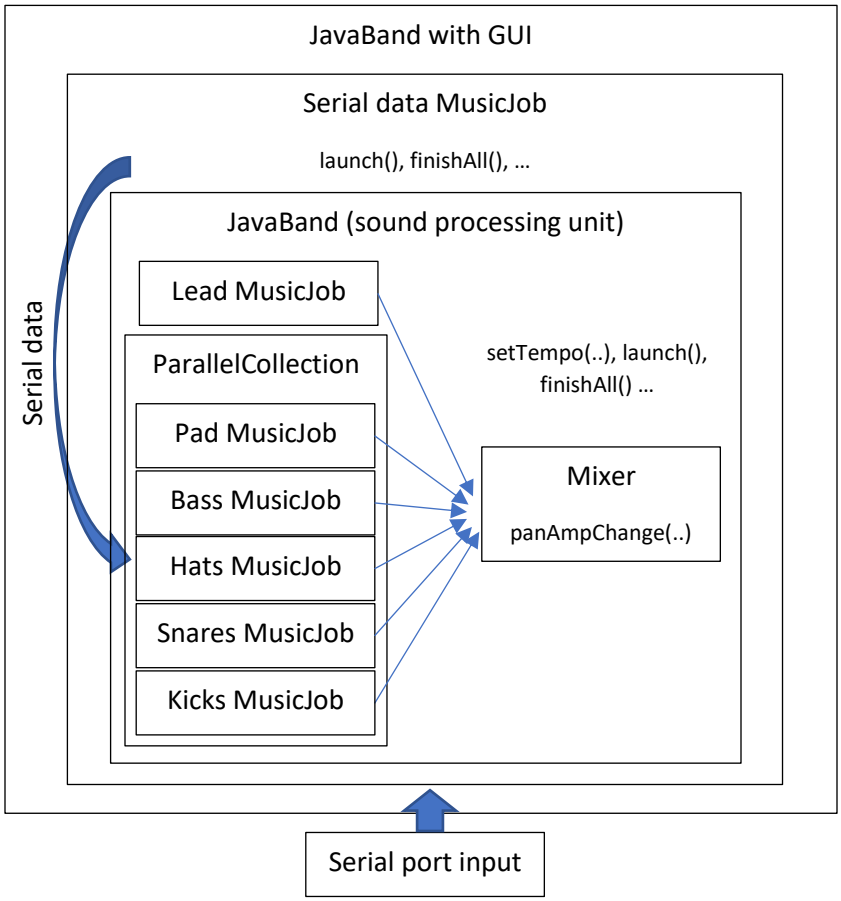

Figure 3. Overall structure of JavaBand

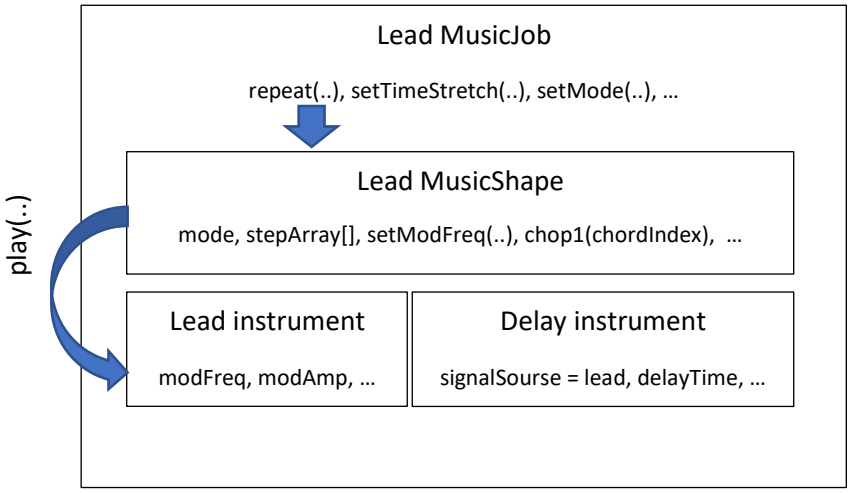

Figure 4. Structure of the MusicJob for the lead instrument

Artistic Analysis

It is extremely exciting to see most features that I have imagined in the conceptualization of this project functioning as I intended, and it is also satisfying to listen to this piece in a musical perspective. Though being a 6-track 1-measure loop, the instruments are carefully designed and orchestrated. The pad, for example, consists of a choir of four voices which forms a major 7^{th} chord with glide feature implemented. The dreamy timbre perfectly suits the chord progression Cmaj7 → Amaj7 → Bbmaj7 → Fmaj7. Meanwhile, the juiciness and beefiness of the lead would balance the music with strong impulses. To make it fun and more danceable, I created a classic 80s' citypop beats for the bass and percussions. Last but not least, the lead uses the same instrument as the bass except that it is a few octaves higher. The phrases are algorithmic composed and artificially triggered by user input, forming an interest harmony between machine and human.

The design of the hardware controller responds to the musical idea as well. The joystick modules remind the player of the arcade games in late 20^{th} century and fits the citypop feel well. The controller is extremely easy and fun to use especially when cueing the lead instrument and adjusting the panning of another instrument.

Shortcomings, however, is also obvious. Most importantly, I have not managed to trigger the lead instrument immediately when pressing the right joystick. Plus, changes to the instruments' parameters would only be applied in the next measure, meaning that there is room for further improving the algorithm. Moreover, it lacks choices for section transitions.

Considering the advantages and shortcomings, a direction for the future works is clear: more musical, more flexible, and more user-experience-oriented. Besides improving the

algorithm as mentioned above, some attempts could be made, for example, applying flexible chord progressions and beats, storing and calling presets for the parameters and different orchestrations, using actual game controllers as well as a recording function.

Conclusion

In this paper, we have discussed topics including the product details, technical and artistic description of JavaBand and pointed out the direction for future works. JavaBand is a hybrid multi-instrumental music player extended on the MusicJob and MusicShape features of JMSL and an Arduino-powered hardware interface. It plays a 6-track pre-programmed musical loop with real-time access to the parameters of the instruments with respect to the concept of computer music synthesis and 80s' citypop music. In the end, we have determined that future works of this project could be made toward better musicalness, flexibility, and user experience. I hope JavaBand could be a start point of my new journey in the path of computer music.