Step Sequencer in Processing

Hugh Rawlinson

Extended Exercise for Introduction to Programming

Academic year 2012-2013

Introduction

For my Intro to Programming Extended Exercise assignment I extended a Step Sequencer sketch I worked on during term 1. The week we were assigned to write a step sequencer was the week before we were introduced to object oriented programming, however having heard of the concept before and having read up on the basic principles I had decided that it would be easier to write something as complex (and with so many repeated but identical components) as a step sequencer in an object oriented style rather than procedurally, as there would have been far too many variables to keep track of, even when using data structures like multidimensional arrays. The step sequencer is extremely object oriented, to the extent that it's essentially a library of co-dependant classes that could be used as components in larger sketches. I tried to write the software in such a way that I would be able to later use each of the classes in a larger project. As a music computing student, I'm sure that they'll come in handy when I inevitably attempt to make some kind of larger sequencer/synthesiser virtual instrument.

Another major component of this software is the ability to save and load patches. During the time I was working on this project, I was lucky enough to be involved in Music Hack Day, a collaborative programming event wherein participants are challenged to make some kind of musical software. The event was hosted at Facebook's offices, therefore there was considerable emphasis placed upon social media, and in particular APIs. I was inspired by the idea of having data in 'the cloud' to be requested on demand by software when the user required it. Having had some experience with PHP, and MySQL, and access to a Raspberry Pi based server in my friends garage in Limerick, I set about designing and building a server-side storage system for sequencer patterns. Given that there are $2^{(8^2)}$ possible pattern combinations on an 8 x 8 grid of binary toggles, I found that it would be easier to store these on a server rather than in a file on a workstation, and given the hegemony and the ever-present internet connection in modern computing, the possibility of the user not being able to connect to the internet was not a huge consideration.

Code

Client Side

All the code on the client side is Processing, however the server to save step-sequencer patterns is written in PHP.

`step_sequencer.pde`

This is the main sketch code to handle initialisation and running of the sketch. It handles the Minim AudioPlayer objects, determining which to trigger at each step, as well as facilitating timing. While visually on the beatIndicator object it appears that the time is running smoothly, the sound does not have a graceful rhythm regardless of input in the sequencer. I spent hours trying to debug this, but after confirming that the audio files had no silence at the beginning causing mis-timing, and increasing the frame rate of the sketch so that `metro.tick` would be called at a higher rate thereby increasing resolution, I was not able to solve the issue. I would speculate that there could be a timing issue with the minim library, possibly while running multiple instances of the AudioPlayer object. Aside from this, `step_sequencer.pde` serves to handle instantiation of the various other classes in the step sequencer, arrange them on the window, and handles user interactions such as clicks and keypressed events which it routes to the appropriate object with the required arguments.

`beatIndicator.pde`

This class is similar to the `track` class that deals with each 'track' in the step sequencer, but is designed instead to be a visual indicator of the current beat that the metro is at. It uses an array of buttons to indicate this, as well as having various properties to facilitate drawing on the window. It has a method `setBeat()` to allow `step_sequencer.pde` to set the appropriate beat with reference to the metro object.

`button.pde`

This class is the fundamental class of the step sequencer - it allows for individual toggle style buttons to be displayed on the screen, with various methods to allow parent objects and sketches to control them by toggling value, setting value, and checking value. The class also allows for ease of display; it has properties `height`, `width`, `xposition` and `yposition` to enable the programmer to draw the button in a similar manner to the `rect()` method native to Processing, but in the context of an object with interactive properties.

`file.pde`

This class facilitates saving and loading from the server. The `writeAndUpload()` method receives a two-dimensional array from the `matrix` object, encodes it in a JSONArray, and sends a HTTP request to a script on a server with the pattern data as an argument, which gets saved into a MySQL database. The `readFromWebsite()` method was slightly more complex to implement. I receive a string in the form of a two-dimensional array from the site which is a representation of a pattern. I had originally intended to read the data from the string into a multidimensional array to send to the `matrix` object to be put into the sequencer, however I had considerable trouble with appending variables of type `int[]` to variables of type `int[][]` despite having previously done so in the `matrix` class, and having used almost identical code. As a workaround, I decided to pass the `matrix` object from the main sketch into this method so that I could call its functions directly without having to deal with concatenating arrays of arrays, and this worked.

Input to (and indeed output from) these methods proved quite challenging. I had tried to use Java's inbuilt Swing library to use `JOptionPane.showMessageDialog()` and `JOptionPane.showInputDialog()` to display IDs to access patterns saved on the server, however I could not get them to show anything other than a blank window without even an 'ok' button, requiring a restart of the software. I did set `writeAndUpload()` to print the ID of the uploaded pattern to the console, but input was trickier. Over the winter break I didn't have time to come up with a more creative solution to the problem of input so unfortunately the ID of the pattern that needs to be loaded must be set in the `arg` variable in the `keyPressed` method of the main sketch. When the key 'l' is pressed, a pattern with ID `arg` is loaded from the server. The default `arg`, 55, is a pattern to demonstrate this feature.

`matrix.pde`
The `matrix` class is an array of type `track[]` with methods and properties to allow it to be drawn with relative ease. Additionally, there are functions to set values of specific buttons, to initialise the values of each button in each of the tracks, to return a two-dimensional array representation of the beats, and to handle clicks on buttons with the help of the `mousePressed()` method in the main sketch.

`metro.pde`
This class is designed very simply to facilitate and gate when the beat is allowed to advance. Each frame the main sketch calls a function in the `metro` object to check if enough time has lapsed since the previous beat to allow the counter to advance. The constructor of this object also allows for tempo to be set.

`sound.pde`
This class is disused, but was originally intended to allow the AudioPlayer objects to be grouped into one object. I abandoned it because of errors caused by the AudioPlayer objects not being in the main sketch.

`step_graphics.pde`
I intended for this class to allow for some graphics ideally to be used on the buttons. Rather than using an "X" shape to denote an 'on' value of the button, I had intended to use an amber glow, inspired by Toby Metcalfe's Traffic Light sketch to emulate the LEDs in Novation's popular grid style MIDI controller the LaunchPad. Unfortunately, there were discrepancies of support between versions of Processing that made having this feature as well as several others run in the same version of Processing impossible.

`track.pde`
The `track` object is essentially an array of `button` objects with a few additional methods and properties to augment ease of use in the context of a step-sequencer. These include methods to set the variables in each of the buttons that deal with drawing the button in the Processing window, to return values of a specific beat in the button array, and to handle a click and toggle the value of the appropriate button.

Server Side

`upload.php`
The upload.php script accepts an argument using `$_GET()` and saves the value to a table in a MySQL database.

`grab.php`
The grab.php script accepts an ID number that corresponds to a value in the table on the server, and returns the data from the table associated with that ID number.

Data Folder

I have included some files to assist in the operation of this sketch. They are 8 samples appropriate for use in a percussion sequencer, including samples of a kick drum, a snare drum, two hi-hat samples, a synth-stab, a crash cymbal, a ride cymbal, and a tom drum. All sounds were taken from freesound.org.

Usage

Upon launch, the sketch will initialise various objects with default values. Values that must be set prior to sketch launch include the constructor variables for the **`matrix`** and **`beatIndicator`** objects that deal with drawing those objects in the Processing window, the constructor variables for the **`metro`** object, which deal with tempo and number of buttons per cycle, and the ID of the pattern to be loaded when a keypress of the 'l' key is detected.

To toggle the value of a button in the sketch between 'on' and 'off', click on the button. The count/beat will cycle automatically and play your pattern as you are constructing it.

To load the pattern with the ID number set at the top of the main sketch, press 'l'.

To save the current pattern to the server, press 's'. The ID number of your pattern will be printed in the console of the Processing IDE.

To initialise the matrix and set the value of each button to 'false', press 'i'.