(M01) Composing with Data Flow Programming
Laura Meskanen-Kundu
Student num. 588247
27.11.2016

Learning Log

Lecture week 5.9 – 9.9.2016

First workshop week and chose to take the Data Flow Programming course to understand and get some idea about pure data (a.k.a Pd). First day was an introduction, second was about control objects, audio manipulation and GEM basics and third about communication, physical interaction, libpd and play with Arduinos. Forth we noted down dynamic processes (compressor, limiters, etc), frequency processes (filters and EQ), time/space processes (delay, reverberation, etc.) and collection of effects, sample players and sound synthesis by Matt Davey. Fifth day was all about Raspberry Pi, patch integrating concepts and our own project pitch. I must say that the first week was a challenge because I wasn't familiar with music theory or terminology, but I tried keeping up best I could. This class has been very difficult to grasp as I've not previously worked with graphic programming environments and with the combination of understanding sound production there was a high learning curve. Pure data seems to be a very handy tool when working with real time sound production. What I did understand was the history of the program having a commercial predecessor, Puckette's original Max program, developed while he was at IRCAM. The software was created to produce interactive computer music and multimedia works.

From the first week lectures I learned about the different types of inputs (object, message, number, symbol and comments). Mostly objects are like functions that are fed different types of data from message (string), number and symbols. Objects have cold and hot inputs where a hot input does an action right away and a cold input does it only when it is called. A special type of message, a bang, initiates the execution of the program flow. This was interesting as a bang is a message with null content. Also, the language houses specialized audio-rate DSP functions (designated by a tilde (~) symbol), such as wavetable oscillators, the Fast Fourier transform (fft~), and a range of standard filters that I'm not yet perfectly clear about, but that are key components in producing electronic music. Overall the initial idea of the entities in pure data was discovered on the first week.

Project week 12.9 – 16.9.2016

The project week started with figuring out how to get started on my project. I decided to go with my strengths and focus on building from there. Because I had an idea in my mind I knew I needed first to be able to access Wikipedia's live stream and from that pass data to Pd. Now how I was going to do this was still a mystery, but there where wiki pages

dedicated to this. I came to realize it wasn't as simple as the example they gave on their pages to access the stream and feed it to Pd for my project.

First struggle was figuring out that an older version (5) of node needed to be installed to be able to access the stream. The newest node version is 7.2. Also as of January 2015, RCStream implements version 0.9 of the Socket.IO protocol, not 1.0 (phab:T68232). For this I needed to read about socket.io 0.9 and socket.io-client 0.9 on GitHub and the documentation. Like previously stated when starting the project, I got access to the stream but getting something from the stream to pure data was a challenge.

To get the everything working I needed to create a local server so that I could have access to the wiki stream from pure data. This is because pure data only supports two ways of accessing external data. One is [netsend] and [netreceive] that are objects for transmitting and receiving messages over a network. The second is Open Sound Control (OSC) and these are objects for sharing musical data over a network. OSC is enabled in most modern programming languages such as Processing, Java, Python, C++, Max/MSP and SuperCollider. In the end, I used a netsend port and send the first letter of the title that we get from the Wikipedia stream.

Next the actual wiki stream access was created so that we can connect it to the opened server port. Here we just connect to the provided RC Stream and when there is a change in data we feed the title of the change.

In Pure Data using the [netsend] and [netreceive] I access the port 8006 to read the data and covert the input to ascii values using the object [spell]. Then with [iter] we split a list of ascii codes into a series of numbers. Depending on this number Pd produced different sounds that where created with the help of using Matt Davey's DIY patches introduced in the first week. To get these patches to work in a personal Pd project one has to copy the necessary patches completely.

Lastly I wanted to connect whatever music was produced into Processing to make graphics out of the live stream data. I could make the connection using OSC, but time ran out to create any cool graphics with the data I was sending with Pd. Also, otherwise I would have wanted to send more data to Pd such as the whole bits received to create more cool outcomes but with my current skill set I wasn't yet able to accomplish that.

A blog post about the project can be found at:

http://lauramk.me/blog/2016/09/17/Pure-data-course/