

Name: James Newhook

Student Number: 1808649

Module Title: Development

Assignment Title: Element 2b

Module Lecturer's Name: Ciarán Robinson and Helen Leigh

Hacker and Maker culture

The definition of hacking, according a number of online dictionaries, features a reference similar to “unauthorized intrusion into a computer or a network” (Techopedia, 2019), which carries negative connotations. This definition stems from the term “hack” which was used by the Massachusetts Institute of Technology (commonly known as MIT) to refer to pranks devised by students of the institute, which in term was adopted by The Tech Model Railroad Club (TMRC) at MIT, founded in 1946, to instead refer to “a project undertaken or a product built not solely to fulfil some constructive goal, but with some wild pleasure taken in mere involvement” that “must be imbued with innovation, style, and technical virtuosity” and thus proudly referred to themselves as “hackers” (Levy, 1984, p. 19). Members of the TMRC would feed data to an IBM 704, a computer “designed for engineering and scientific calculations” (IBM 2019), in an attempt to “hack” the machine to discover what capabilities it had further to its intended purpose. This “hacker culture” that was based on curiosity and advancing technology, is different from the common and negative definition above that a lot of people and the media associate with the words “hack”, “hacker” and “hacking”. The hacker community actually refers to these so-called “hackers” as “crackers” and do not wish to be associated with them (Raymond, 2001).

The hacker culture of today is very community and sharing based with websites such as GitHub (github.com) which is the largest code host in world (Gousios, G et al. 2014) and the Cycling ‘74 forums (cycling74.com/forums/page/1) where the community helps those using the visual programming language, Max. There are also events called Hackathons, where “people work in groups on software or hardware projects, with the goal of creating a functioning product by the end of the event” (Dictionary.com, 2019).

This community and collaborative spirit is also seen in the maker culture which, while often used as a synonym for hacker culture, refers more to physical “makes” compared to digital and computer based “hacks”. In his book, *Makers: The New Industrial Revolution*, Chris Anderson talks about how the increased availability of the tools required to make “makes” to the consumer, and how modern production techniques and a community based on sharing skills and knowledge, has helped spark the maker movement (Anderson, 2012). Makerspaces, which are social community workshops (South London Makerspace, 2019), further fuel this movement by providing makers with access to tools they might not be able to afford or have space for in their homes.

The maker movement has produced makes ranging from pocket radiation detectors designed in the wake of the 2011 Japanese nuclear disaster to real vegan cheese made by engineering yeast (Martiniere, 2014).

Tools and production methods

When planning a make or a hack, there are a number of things to consider. The first of these is if the final product will be physical or digital, or a combination of both. Physical makes commonly feature a single-board computer such as those produced by the Raspberry Pi Foundation. Raspberry Pi computers runs the operating system, “Raspbian” which is “a free operating system based on Debian optimized for the Raspberry Pi hardware” (Raspbian.org, 2019). This is an open-sourced ecosystem (Opensource.com, 2019) which is ideal for makers as it saves money, is better for making tech and is improved by collaboration from its community (Pearce, 2016), something that it shares in common with the maker movement itself. The Raspberry Pi, along with other single-board computers such as those made by Arduino, can connect to a range of devices such as ultrasonic transducer modules, infra-red sensors and LCD displays, making them versatile tools for creating products (Allan, 2013). You can then program these computers to carry out the functions you require.

For products in the digital realm, there are a plethora of programming languages and environments that can be used and which is best will be depend on the type of product being made. For music and sound applications, some common tools are Max, Pure Data and SuperCollider. Each have their pros and cons and one may be chosen over the other depending on the application of the product.

SuperCollider is an open-source scripting language and environment for audio synthesis and algorithmic composition. It uses “sclang” which is an interpreted programming language (SuperCollider, 2019). SuperCollider has been used to create products such as generative music devices (Trunks7j, 2009) and visual synthesizers (Murthy, 2017).

Unlike SuperCollider, Max and Pure Data (Pd) are both visual programming environments, rather than scripting based. Both Max and Pd were originally authored by Miller Puckette and as a result are very similar applications (Conners, 2019). One of the key differences between Max and Pd is that Pd is free and open-source whereas Max is owned and sold by Cycling '74. Pd is more versatile for commercial production as it “can incorporate objects that have been developed using other languages, such as C, Python, and Scheme” (Conners,

2019). For physical products, you can code directly onto single-board computers such as the Bela Board (Bela, 2019) using Pd, which is not possible with Max (this is however, possible with SuperCollider). However, Max is paid for program and therefore comes with its bonuses such as regular updates and better documentation and tutorials (Cycling '74, 2019).

For commercial distribution of instruments and devices made in Pd, there are other options in addition to the Bela board mentioned above:

- Unlike Max it is possible to turn any Pd patch into an iOS app for iPhone and iPad (Kirn, P, 2016).
- To create a plugin that can be used within a Digital Audio Workstation (DAWs) or other audio application, there are tools such as PdVst which allows Pd patches to run as a VST plugin (Pure Data, 2018).

Conversely, Max discontinued VST support in 2009, instead focussing on Max for Live (M4L) which is a version of Max that operates within the DAW Ableton Live (Cycling '74, 2009). Max for Live comes packaged with Ableton Live Suite or can be purchased as an add-on to other versions of Ableton Live (Ableton, 2019). Max for Live's integration with Ableton Live does mean that it is quick and easy to develop a graphical user interface (GUI) that can be tested in a DAW. This is a useful feature for prototyping a plugin that could later be redeveloped in another programming environment for commercial distribution.

Considering the pros and cons of these three platforms, I have summarized who and for which application each might be best for:

- SuperCollider would be suited to someone who already has experience with script-based coding. Given the examples shown above, it is ideal for interactive products and for use on single-board computers such as the Bela board.
- Pure Data, with its visual programming environment, would suit someone who has little or no script-based coding experience but who may be a music or audio specialist that is used to the connection of cables and flow of signal. It is also ideal for commercial distribution as it is easily exported to a number of user-friendly applications such as VST plugins and iOS apps.
- Max, similarly to Pd, is ideal for music and audio specialists. It's well integrated tutorials and help files make it suitable for beginners and, if using Max for Live, its seamless integration with Ableton Live means that it is easy for the creator of the patch to quickly apply its functionality with a DAW.

Instrument conceptualisation and market research

Rather than conceptualise my instrument and then choose the best platform on which to build it, I instead chose the platform first. My reasoning being that I was interested in the possibilities of Max as a live music and sound tool, and how it's integration with Ableton Live could open up the possibilities of my live performances. I felt that this project would be a great way to begin learning Max and, from comparing it to other programming environments such as SuperCollider and Pd, I felt it was the best for beginners to programming (like myself) given its tutorials, help documentation and strong community.

One of my first ideas was to create a sampler that produced pad sounds. The idea was that you could load samples from your song and then create pad sounds based on them. The user would feed it a chord sequence, which it would analyse and then create evolving but subtle soundscapes based on which chords were being played. When considering the logic involved in calculating which notes to play based on a changing chord sequence, I felt that this a very difficult project for a beginner.

Continuing the idea of an instrument that takes user samples and turns them into something else, I looked into generative music which is "a term popularized by Brian Eno to describe music that is ever-different and changing, and that is created by a system" ('Generative Music', 2019). My initial idea was that you could load a number of samples, or stems from a song into the instrument which could then be played, with the result being something that sounds different yet familiar. This was in part inspired by a concert I attended by the band Silversun Pickups (2016), where, for around 20 minutes prior to them coming on stage, an evolving soundscape that sounded similar to the intro of their first song, played continuously until they came on stage at which point they seamlessly began playing the song in sync with the soundscape. I thought that a device like this could be used to create familiar and cohesive sounding background music for live performances as well as continuing my previous idea of a pad generator, albeit simpler.

When searching for similar existing products I came across the Siren Audio Generative 2 (Siren Audio, 2019) which is a generative music application based on granular synthesis. I was unable to find the location to purchase this instrument and came to the conclusion that it was no longer available. This was backed up by a Google search that returned results dated no later than 2016 and the Siren Audio blog's last post which was in January 2018. A review of the application on the MusicRadar website (Computer Music, 2015) explains that it relies on various random factors to create abstract, ambient soundscapes. Once the parameters

are set and the application is activated, there is little user input and instead you are to rely on the instrument's randomness to create evolving soundscapes. The review criticised its insubstantial documentation and convoluted interface.

Another existing product is Cataract by Glitchmachines (Glitchmachines 2019) which is termed by the company a "Segment Multiplexer". Compared to the Generative 2, Cataract is more capable of producing glitch sounds as it is not based on granular synthesis. Cataract comes with its own samples but you are able to load your own, and you can randomly select these samples from file. Every parameter in the application can be randomized and it includes effects like the Generative 2.

Something that both of these instruments have in common is that they are completely in charge of generating their output once the parameters have been set and cannot be played by a MIDI keyboard or MIDI sequencing. I therefore decided that this would be a key factor in making my instrument different to those already available.

Instrument Pre-Production

My final pre-production idea was a playable sampler that featured a number of randomizable features to allow the creation of generative music.

Before beginning to plan how I would build my instrument, I first completed some of the inbuilt Max tutorials to give myself a basic understanding of how the program works and the workflow needed when creating a Max device. I then created an overview of how the instrument would work, detailed in the diagram below.

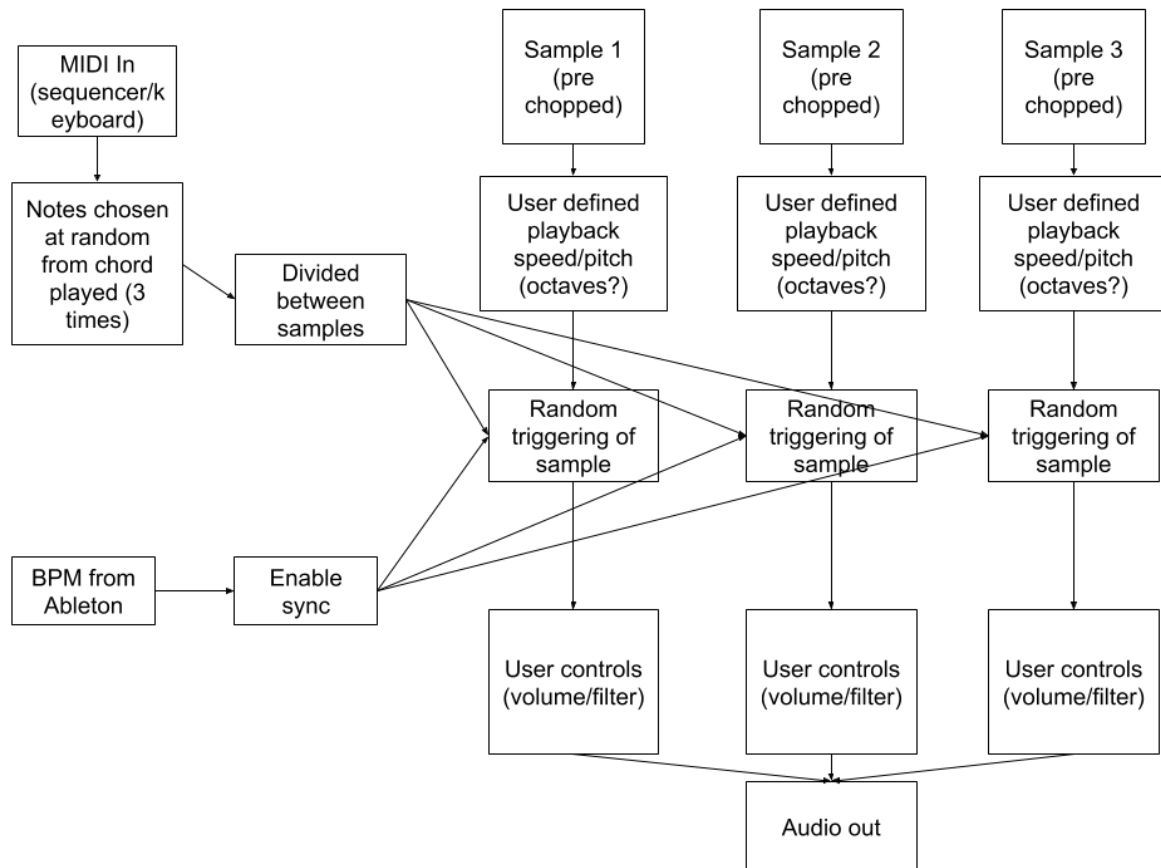


Figure 1 Instrument overview diagram

I decided to start by building a basic sampler in Max that I could then base my instrument around. From some tutorial sessions I'd had with sound artist and Max user, James Wilkie, I had come to understand that there were always multiple ways you could achieve something in Max with the various objects.

Objects in Max carry out functions ranging from simple mathematical calculations to switches and sound wave generators. The best way to investigate an objects capability is by opening and exploring its help patch. From this point you can also find discover other objects with similar functions by accessing the question mark tab within the help patch.

Production

When building my sampler I looked into three different objects; groove, play and wave. Of these I felt that groove would be the best on which to base my sampler as it featured looping and variable rate playback. My first attempt to build a sampler is in the image below.

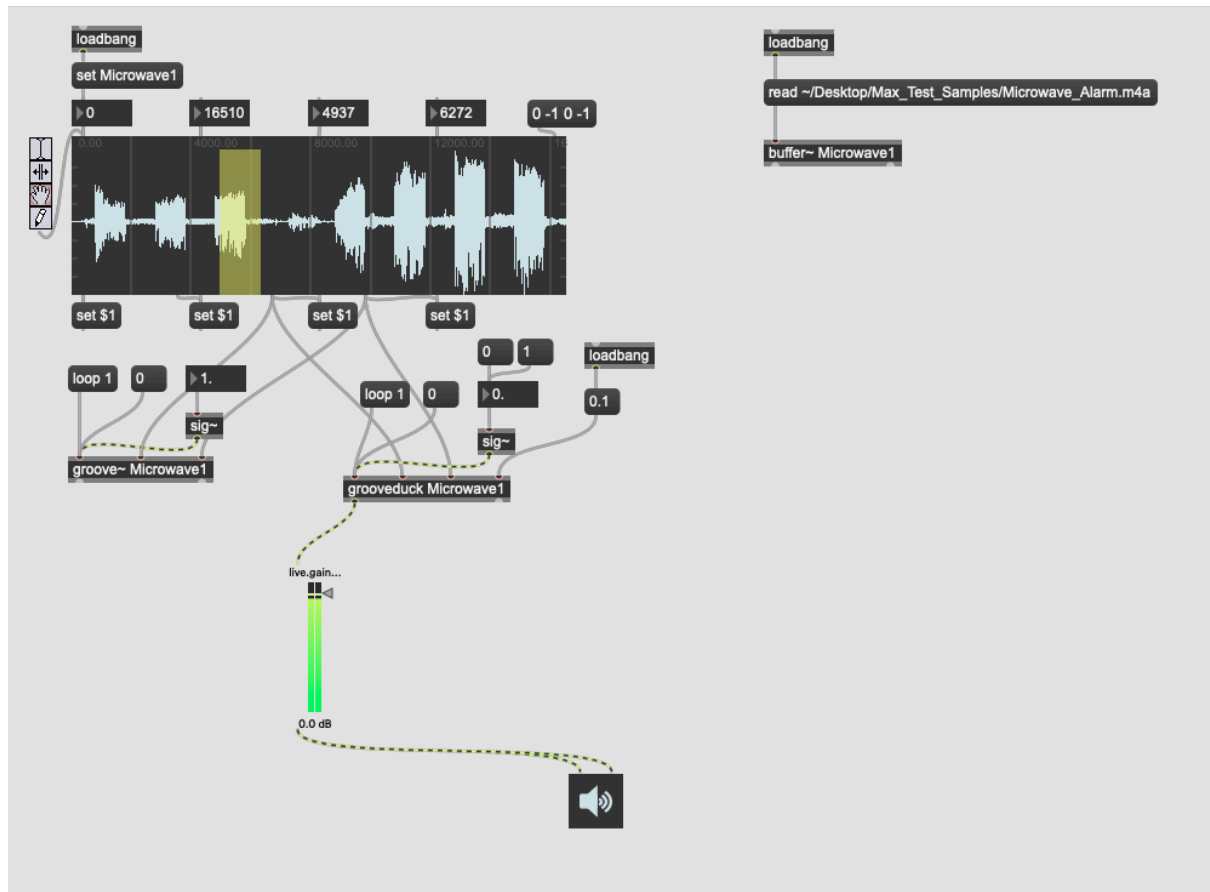


Figure 2 My first sampler

This sampler was not yet playable with a keyboard but had loop functionality and playback rate options. I also implemented the waveform object, which, in its help patch, provides a toolbar that can be used for sample selection and zoom functions. This was a user friendly and interactive feature I felt that my sampler would benefit from. In the image there is both the groove and grooveduck objects implemented. The grooveduck object is the same in every way as the groove object except that it applies a short fade to the end of the sample to resolve issues with audible clicks, which I had been having with the groove object.

My next step was to research how to transpose and pitch shift the sample in order to play it with a MIDI keyboard. When researching this I used both the Cycling '74 forums and the website maxforlive.com to download existing examples of samplers with pitch shifting and

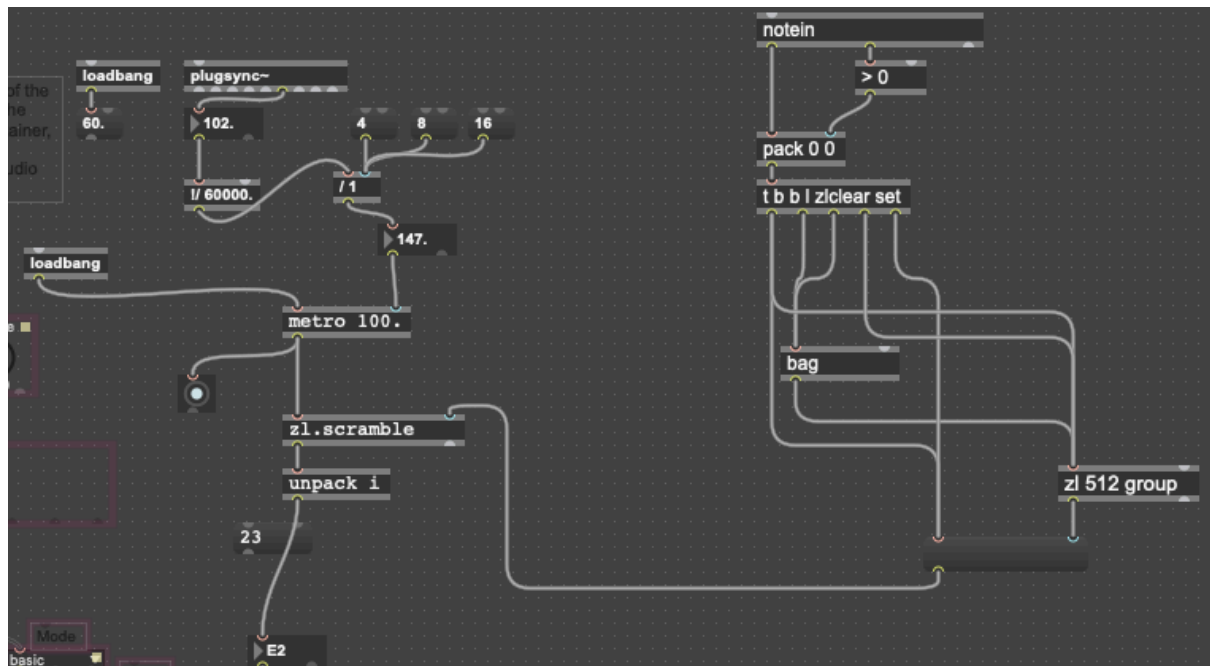


Figure 5 Note randomizer

I then put my randomizer into a subpatcher object, which allows one patch to live inside another. In the subpatcher I expanded on the randomizer and tidied it up. When testing my instrument at this point, I continued to have issues with the notes being held, especially when I integrated a key range function which allows the user to select which inputted notes the instrument responds to. First, I had to sacrifice the variable velocity of the instrument by instead triggering notes using the 'notes held' information outputted by the borax object. I then needed to find a way to stop notes outside the specified key range from triggering the last note output by the randomizer. I eventually removed the fix in the image below (Figure 6) as I found that by just connecting the borax object to the split object instead of direct to the note in, the issue was solved.

Reflecting on the number of fixes needed to make the note randomizer work, I feel that part of the issue was me not considering that there are two messages sent for each note; a note on and note off. With this in mind, there may be a simpler way to program the randomizer which in turn could reduce the processing requirements of the instrument.

Another area I added was the note length selector and randomizer. I have seen other patches and examples where note length does not require the calculations I have used. This is another area that could be simplified.

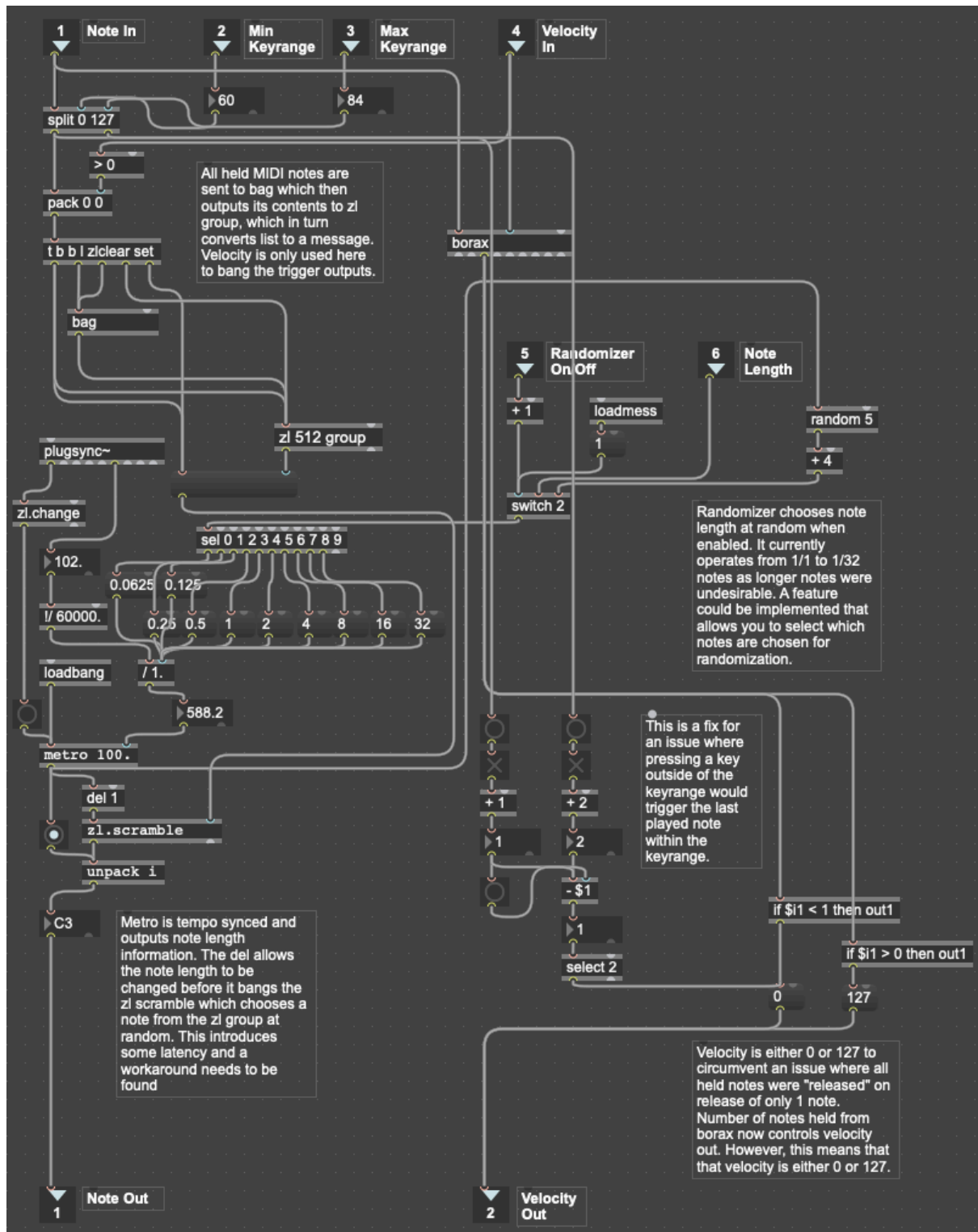


Figure 6 Randomizer subpatch

I next addressed the GUI which was still designed for the More Simpler device. I kept a lot of the functionality of that GUI but added my features and removed the loop, reverse, snap and velocity sensitivity parameters as these either no longer worked or were redundant.

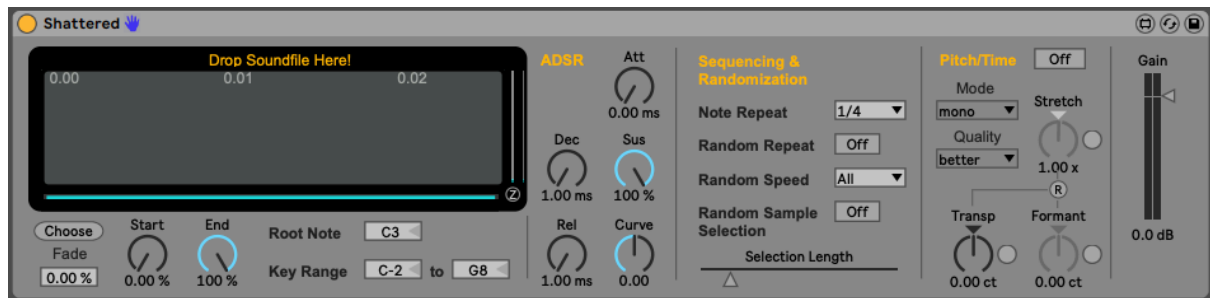


Figure 7 My GUI

I returned to my initial diagram (Figure 2) and felt that I'd achieved most of what I'd set out to do. I did some brief user testing however, and the feedback I received was that the randomization features were comparable to the random setting on some arpeggiators. With this in mind, I looked at what I could do to add further randomization and take a step further towards generative music. I realised a simple amendment was to add randomization to the sample selection parameters. To do this, I just used the existing note length information output by the metro object in the note randomizer to feed another random object that decides the sample starting position. Another control determines how long the sample will be (this is not random). This was simple to implement, however there is an issue with the selection length, where the Selection subpatcher from the More Simpler device sometimes does not change the selection end when instructed to. I believe this could be as I am controlling the sample length by the dials on the GUI, rather than going direct into the Selection subpatcher which does not have time to make its calculation.

Reflection on my instrument

With this new feature implemented I feel that my instrument has become what I intended it to be with an extra level of randomization that adds to its generative music attributes. The 2 examples of music I have included with my submission, and that are detailed in the user guide for the instrument, display both a variety of sounds the instrument is capable of as well as 2 different ways of operating the instrument to achieve different results.

There was only one key area that I did not achieve which was to create multiple instances of the sampler in one Max patch. When I attempted this I found that there was a lot of renaming and redirecting of objects required, something I did not have time to learn and implement. I would also need to create a mixer section to make this a worthwhile feature.

There are some outstanding issues with the instrument as follows:

- There is often still an audible click when a sample is retriggered, an issue that wasn't fully solved by the implementation of the grooveduck object or through changing the envelope settings of the instrument. I anticipate that a fade in and out of the sample needs to be triggered by the note length.
- It is currently not possible to circumvent the random note selector. Although this is a key part of the instrument, being able to disable it would lead to more uses of the instrument.
- Currently, pressing a key doesn't reset the synchronisation info within the patch. This means that if you choose 4 bars as your note length, you may have to wait up to 4 bars to hear a sound once you've pressed the key.
- There is no velocity variation since I had to remove this to solve some issues.
- The random sample selection issue mentioned previously.

In comparison to the two instruments I looked at before beginning production of my instrument, I feel that my instrument is quite basic in its generative music capabilities and I'd consider it to be more of a semi-generative instrument. I do however, believe this is in some ways an advantage as one of the issues reported with Siren Audio Generative 2 was that its interface was unintuitive. Cataract also has a large amount of controls and functionality that can be intimidating, compared to my instrument which is fairly streamlined. However, I do feel that the implementation of more features would improve the usability and create more applications of my instrument.

User feedback

(Please see appendix i for my notes taken from the user testing session)

When conducting user testing, one of the main areas of feedback was that this is a product only available to those using Max or M4L. As discussed previously, to turn my instrument into a commercial product it would need to be rebuild it using a different programming environment, although Max is an excellent platform for prototyping. Now that I have some experience with Max I could look into Pure Data as it is much easier to port a patch into a VST that can be used in more than one DAW.

In terms of the functionality of the product, I received feedback that the formant control acts more like a filter or tone control. Based on this I would likely remove the formant function and implement a fully functional filter; something that is common in a lot of sampler instruments.

One user wanted to know if the random sample selector snapped to transients. This is a feature that could improve the appeal of the instrument as it would make the music it outputs less abstract. This is a feature that worked in the original More Simpler patch which would be a good reference on how to implement this.

Another user was curious about what happens when you load a whole, mixed song into the sampler. This is something I will test and possibly produce an example which can be used when demonstrating what the instrument can do.

Future of my instrument

I would first like to resolve the issues the instrument has and implement the feedback received from the user testing before testing again with people who regularly use Max and M4L. I would also conduct more detailed market research to see if there is space in the market for a product like this. Otherwise, I would continue developing the instrument to suit my needs in order to further develop my sonic palette and signature sound which in turn could separate me from the crowd and lead to more work and success in my career.

Bibliography

Techopedia (2019) *Hacking*. Available at:

<https://www.techopedia.com/definition/26361/hacking> (Accessed: 5 April 2019).

Levy, S. (1984). *Hackers: Heroes of the Computer Revolution*. New York, New York, USA: Dell Publishing.

IBM (2019) *704 Data Processing System*. Available at:

https://www.ibm.com/ibm/history/exhibits/mainframe/mainframe_PP704.html (Accessed: 5 April 2019).

Raymond, E.S. (2001) *How To Become A Hacker*. Available at:

http://catb.org/~esr/faqs/hacker-howto.html#what_is (Accessed: 5 April 2019).

Gousios, G et al. (2014) *Lean GHTorrent: GitHub Data on Demand*. 11th Working Conference on Mining Software Repositories, MSR 2014 - Proceedings, pp. 384-387.

Dictionary.com (2019) *Hackathon*. Available at:

<https://www.dictionary.com/browse/hackathon> (Accessed: 5 April 2019).

Anderson, C. (2012) *Makers: The New Industrial Revolution*. London, UK: Random House Business Books.

South London Makerspace (2019) *South London Makerspace*. Available at:

<https://southlondonmakerspace.org/> (Accessed: 5 April 2019).

Martiniere, M (2014) *Hack the World: How the Maker Movement is Impacting Innovation*.

Available at: <https://medium.com/@mmartiniere/hack-the-world-how-the-maker-movement-is-impacting-innovation-bbc0b46bd820> (Accessed: 5 April 2019).

Raspbian.org (2019) *Welcome to Raspbian*. Available at: <https://www.raspbian.org/> (Accessed: 5 April 2019).

Opensource.com (2019) *What is a Raspberry Pi?* Available at:

<https://opensource.com/resources/raspberry-pi> (Accessed: 6 April 2019).

Pearce, J. (2016) *How maker communities align with open source* Available at:

<https://opensource.com/life/16/11/maker-open-source-communities> (Accessed: 6 April 2019).

Allan, A. (2013) *10 Things to Connect to Your Raspberry Pi*. Available at:

<https://makezine.com/2013/04/11/10-things-to-connect-to-your-raspberry-pi/> (Accessed: 6 April 2019).

SuperCollider (2019) *SuperCollider*. Available at: <https://supercollider.github.io/> (Accessed: 6 April 2019).

Trunks7j (2009) *Generative music in Supercollider & Processing*. Available at:

<https://www.youtube.com/watch?v=rMbcqv8rxnA> (Accessed: 56 April 2019).

Murthy, A. (2017) *VisualSynth - A Processing + SuperCollider project (Hands On)*. Available at: https://www.youtube.com/watch?v=BnRpOu-Y_7Y (Accessed: 6 April 2019).

Conners, J. (2019) *Pure-Data (Pd) Programming: How to Get Started*. Available at: <https://www.whoishostingthis.com/resources/pure-data/> (Accessed: 6 April 2019).

Bela (2019) *What is Bela?* <https://bela.io/about> (Accessed: 6 April 2019).

Cycling '74 (2019) *Older Versions of Max/MSP/Jitter and Cyclops*. Available at: <https://cycling74.com/downloads/older> (Accessed: 6 April 2019).

Pure Data (2018) *How To Install PdVst*. Available at: <https://puredata.info/docs/HowToInstallPdVst/> (Accessed: 6 April 2019).

Cycling '74 (2009) *Pluggo Technology Moves to Max for Live*. Available at: <https://cycling74.com/newsletters/pluggo-technology-moves-to-max-for-live> (Accessed: 6 April 2019).

Kirn, P (2016) *There's a new way to make your iPhone run any Pd patch, free*. Available at: <http://cdm.link/2016/11/theres-now-way-turn-iphone-pd-patch-free/> (Accessed: 6 April 2019).

Ableton (2019) *Live*. Available at: <https://www.ableton.com/en/shop/live/> (Accessed: 6 April 2019).

'Generative Music' (2019) *Wikipedia*. Available at: https://en.wikipedia.org/wiki/Generative_music (Accessed: 7 April 2019).

Silversun Pickups (2016) *UK and European Shows*. [O2 Academy Islington, UK. 2 March.]

Siren Audio (2019) *Siren Audio*. Available at: <https://sirenaudio.co.uk/> (Accessed: 7 April 2019).

Computer Music (2015) *Siren Audio Generative 2*. Available at: <https://www.musicradar.com/reviews/tech/siren-audio-generative-2-624182> (Accessed: 7 April 2019).

Glitchmachines (2019) *Cataract: Segment Multiplexer*. Available at: <https://glitchmachines.com/products/ataract/> (Accessed: 7 April 2019).

Appendices

Appendix i, user testing feedback

