

Tileyard Education

MA Commercial Music Producer

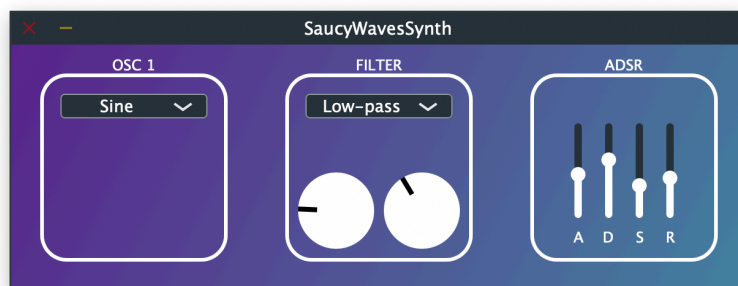
Term 2 - Development Assessment

ADAD7005_LON

Aurélien Merz

Student ID: 1808640

Tutor: Helen Leigh



Introduction	3
Agenda	3
Analysis	3
Technical Details	5
Approach & Practical Implementation	6
Results & Testing	9
Encountered Issues	11
Comparison & User Feedback	12
Improvements & Vision	12
Conclusion	14
References	14
LINKS	14
List of Figures	14
List of Tables	15

Introduction

Making music has evolved dramatically since the time when tapes and the first sampler were making their debut. Electronic components with better quality and performance helped to develop new instruments where musicians or music producers could express themselves quicker and especially with new sounds. Synthesisers and samplers changed the way music was made and remains to do so. In the digital era, computers have taken over the analog instruments by emulating them and by sometimes going several steps further with the arrival of the MIDI standard.

The goal of this second term project was to develop a software or hardware synthesiser using some of the knowledge acquired during the Creative Music Technology week and to implement a first working prototype.

Agenda

The agenda for this second term assessment is the following:

1. Choose between making a software or hardware instrument or synthesiser
2. Make an analysis and market research for that particular instrument
3. Implement the instrument and test it
4. Discuss results and improvements

Analysis

1. From Abbey Road to the bedroom

Back in the day, making music and delivering it as a product was a lengthy process. Artists had to go to studios equipped with high-end and expensive analog gear like pre-amps and compressors to produce a record. This process involved significant production costs and was also very time consuming. Although some artists and producers still go to studios nowadays, homemade music has had a significant impact on how music is created. It became easier and cheaper for producers and artists to work from their home and not need to be in a proper studio in order to create and deliver quality products. The number of bedroom producers number increased substantially during the last decade. The explanation of this phenomena is that bedroom producers / artists could install a variety of plug-ins on their machines, such as a synthesiser or audio effects replications, that would help them make music with just a laptop. The strong augmentation of powerful music applications and software in parallel with computers integrating new high performance CPUs each year is also a major cause for this.



Photo by [John Hult](#) on [Unsplash](#)

Furthermore, the internet remains to be a big influence as services like YouTube helps propagating knowledge.

2. Competition

Nowadays, it is not uncommon to hear that big hits have been made solely by use of synthesiser plug-ins in the form of software plug-ins as it has increasingly become a significant part of music creation and the music business in general. Sound engineers, artists and producers could not always afford to purchase some analog gear and equipment such as pre amps, equalisers and compressors, for example, so buying replications of famous synths or audio effects became more attractive. This market has become very profitable and many hardware audio signal processing manufacturers started making emulations of their hardware products in order to fit into that particular market like Universal Audio, for example. New companies were born from making synthesisers plug-in only as well, making them specialised in that field.

Competition has always been a major thing in any businesses and especially during the last two decades where the improvement of softwares has reached exceptional outcomes. Even if some producers or artists prefer the warmth of analog synths and prefer to buy the real instruments, this is not something the everyday musician can afford. Emulations of famous synths such as the Roland Juno 106 or Sequential Prophet 6 have reached remarkable similarities to their analog peers and for a much more affordable price than actually purchasing the real one.

3. Influence of the internet

The development of the internet has drastically changed the way people buy and consume products and how manufacturers have been able to efficiently target their customers. While the internet has made it easy for customers to browse through the overwhelming world of software plug-ins, it has also inevitably had some negative side effects to this market. Piracy was the main reason why plug-ins and DAWs increased their selling price and it is still a major issue nowadays. Even though plug-in prices were relatively cheap compared to their analog siblings, people were still illegally downloading the products without realising they would miss on customer services and potentially free updates from the manufacturers.

As a strike-back to this crisis, some companies started introducing new ways for musicians and producers to purchase high quality plug-ins and softwares without noticing payments. This method arrived under the label "rent-to-own" and was introduced by famous companies such as Splice. This helped thousands of people to rent expensive plug-ins by paying a fixed monthly amount between \$9 to \$12, which eventually lead to full ownership of the plug-ins after the full amount was paid. Manufacturers also developed their own online stores where customers could purchase the replication of their products with discounts or special offers and keep them in the cloud.

4. Product Description

Saucy Waves is a polyphonic midi plug-in synth influenced by other known synths such as Native Instruments **Massive**, Xfer **Serum** or ROLI **Equator** to name a few. It generates different sounds using three different oscillators and includes two units: a filter unit where the user can choose between a lowpass, high pass and band pass filter and an envelope unit with 4 controls: Attack, Decay, Sustain and Release also known as ADSR.

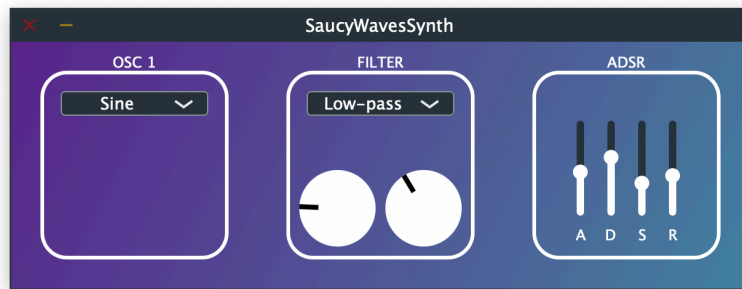


Figure 1: Saucy Waves plug-in v1.

Saucy Waves is designed for any musician, composer, producer, sound designer or artist looking to create unique sounds that will define their sound. This product supports VST3 and AU and is compatible with any operating system available on the market.

Technical Details

This section serves to provide some context as it relates to clarity regarding what the product does and how it functions. Audio plug-ins are little software applications that can be “plugged in” to DAWs in order to reinforce their usage potential and functionality. Audio plug-ins are mostly digital audio processing (DSP) units or sound synthesis and often include user-friendly UI. There are three main categories of plug-ins also known as Steinberg’s Virtual Studio Technology (VST):

1. **VST Instruments:** Plugins that generate audio like synthesizers or samplers
2. **VST Effects:** Audio processing blocks such as EQs or compressors
3. **VST Midi Effects:** Processes MIDI messages

Apple has developed their own plug-in version called Audio Unit (AU) that is made to enhance performance on iOS and macOS platforms. VST can also work on Apple’s operating systems, as a matter of fact.

In practice, DAWs like Ableton, Logic Pro or ProTools will act as the host for the plug-in where blocks of audio data are exchanged between the two. The process is fairly simple as the host will send a block of audio data to the plug-in and will receive back a new block of processed audio data as an output. Audio blocks can have different sizes and are mainly a power of 2 such as 1024, 512 or 128. The size of these blocks will influence the latency of the host in processing audio and outputting it.

As mentioned previously, some plug-ins will produce sounds themselves. These types of plug-ins are most commonly stimulated by MIDI inputs produced by MIDI controllers or keyboards. The software plug-in for this term project is a midi controlled synth coded in C++14.

Approach & Practical Implementation

1. Tools

This section aims to describe the development process necessary for the creation of the instruments and the subsequent implementations. There were a variety of tools that were used in the development, testing and execution of this software plug-in, which are each listed below:

- IDE: Xcode 10.1
- Projucer: JUCE's project management and generation
- Language: C++14
- Framework: JUCE 5.4
- [Maximilian DSP Library](#)
- [GitHub](#)
- Operating System: macOS Mojave
- Novation Launchkey 25 MIDI keyboard
- Logic Pro X

In order to build such an audio plug-in, two libraries have been used: JUCE and Maximilian. JUCE is an open-source but also subscription-based cross-platform C++ framework developed by English music company ROLI. It is mainly used to create desktop and mobile audio applications and GUIs. Maximilian is also an open-source C++ library specifically used for audio signal processing and sound synthesis.

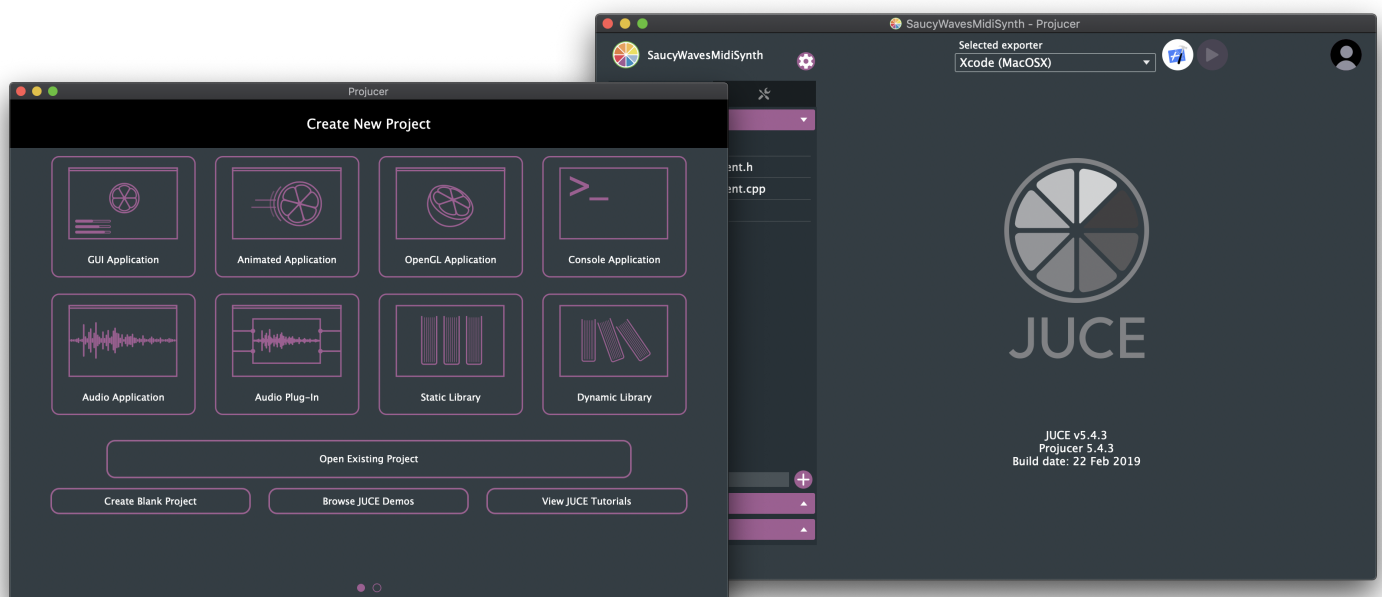


Figure 2 : Projucer desktop app

The Projucer is an application provided by JUCE to help set up the right configurations for the different set of applications such as audio application, audio plug-in, GUI application and many more. Once the project is correctly configured with the required libraries, the user can save the project and the Projucer will open the user's IDE.

2. Software Architecture

The architecture of the entire project is based on object oriented programming (OOP) and is written in C++14. This type of programming is not sequential, but rather multithreaded which means that a central processing unit, which is commonly known as a CPU, can execute more than one process or piece of code in parallel. This is a very important point in digital audio processing and one of the most critical as well because audio applications require real-time computing¹.

The main libraries used are the Maximilian for audio processing and the JUCE Modules for all the GUI, handling the MIDI messages and the synthesiser itself. The Maximilian framework was used to create the oscillator and filter objects². The entire plug-in consists of an ensemble of classes defining the software architecture:

a. The PluginProcessor Class³:

This class represents the entire plug-in itself and handles most of the configurations and initialisation of the other classes. It sets all the connections from the host to the plug-in and also handles the MIDI events and messages.

b. PluginEditor Class:

The PluginEditor class acts as a container for GUIs. Each class has its own GUI and the PluginEditor makes them visible to the user. It also handles MIDI messages.

c. SynthVoice Class:

A synthesiser contains voices that can play one sound at a time. In order to be polyphonic, it has to contain multiple voices to play multiple sounds. If it only contains one voice then it would be monophonic. This class is where the rendering of audio is done and where the sounds are produced and outputted. It handles the MIDI notes played and converts them into their corresponding frequency so the oscillators can play. It uses the next class described below.

Inside this class reside the main objects or components of the synth such as the oscillators, the filters and the envelope.

¹ Real Time Computing describes systems that are time sensitive and that must perform tasks under a specific period of time.

² In OOP an object is an instance of a class. It contains data and functionality defined in a class.

³ In OOP, a class is the definition of some data, functionality that will dictate the behaviour of an object.

d. SynthSound Class:

This is a passive class that inherits from SynthesiserSound class and describes what the sound is. A sound can be triggered by one or more midi notes or by a channel.

e. Oscillator Class:

This class contains all the GUI information for the oscillator section of the plug-in like the drop down menu to select which oscillator to use. The user can select between three different oscillators from the Maximilian framework: Sine wave, saw wave and a square wave.

f. Envelope Class:

This class defines the GUI elements that will shape the sound of the synth. It contains four sliders that make the envelope of the sound. An envelope is defined by special parameters measured in milliseconds:

- **Attack** is the time taken for the sound to go from zero to max level from when the key is pressed
- **Decay** is the time taken from the attack level to the sustain level
- **Sustain** the sound's duration until the key is released
- **Release** is the time between the sustain level to zero after the key is released

The sustain is measured by level instead of milliseconds. The fader values will be propagated to the SynthVoice class where they will be used by an envelope object from the Maximilian framework again.

g. Filter Class:

The Filter class also has faders as GUI elements as well as a drop down menu to choose between a low pass, high pass and band pass filter. The two fader values are the cutoff frequency and the resonance of the filter itself and they will also be passed to the SynthVoice class.

3. Inputs & Outputs

This plug-in is stimulated by MIDI inputs coming from any MIDI keyboard or from the host directly. The plug-in handles the MIDI messages and converts the notes played on the keyboard into frequencies that the synthesiser can process and play back to the host. Depending on the selection on the oscillator menu, the frequency will be given to the selected oscillators and will play the sound.

In the figure, the plug-in has three main connections: the MIDI connection from the MIDI

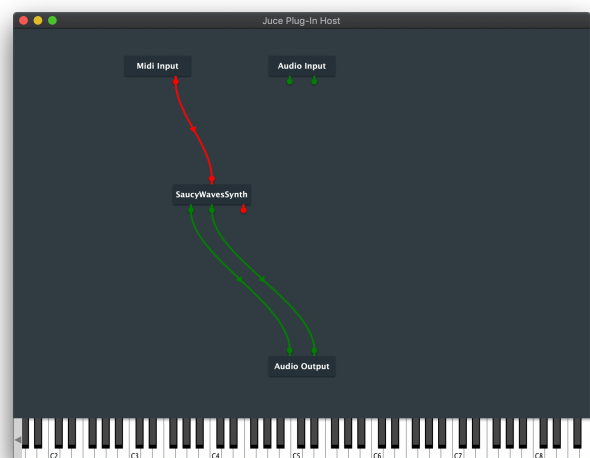


Figure 3: Saucy Waves plug-in inside the host simulation.

input block and the stereo left and right audio connection to the audio output.

Results & Testing

In software engineering there are different levels and types of testing that are taken throughout the whole development cycle and life cycle of a software application. Generally speaking, testing is known as the method of applying inputs to a piece of code and comparing the resulting output with the expected output based on robust mathematical models from Matlab for instance. The most common testing types are:

1. **Unit testing**, which is the process of performing a number of tests on a particular unit itself to check whether or not it is error free.
2. **Integration testing**, which is the process of testing if all units are working in conjunction with each other when all integrated together.
3. **System testing**, which occurs when the entirety of the code is compiled as a product and is subsequently tested as a whole before being sent over to the customer.
4. **Regression testing**, which is the automated testing of a product after it has had some maintenance code updates or new features added to ensure that these new changes have not affected the original, expected product behaviour.

For this project most of the testing has been done following the first three testing types described previously, as the regression testing requires several specific resources that were not readily available; furthermore, and most importantly, as the project is still in a prototype stage, it has not been subject to enough code updates or feature additions to merit regression testing.

Another point to mention is that most of the testing has been done manually. Usually software should also be stressed at the unit level by the use of automated testing as this is a much faster method and generally guarantees better testing. It is also the most reliable method for identifying any existing edge cases and revealing new issues that may have otherwise remained unknown. This is usually done by running scripts written in Python, for example.

That said, below is an overall summary of the manual tests performed, illustrating which tests passed and which tests failed.

TESTS	PASSED	FAILED
Instantiation of plug-in		
MIDI Messages handling		
Sound output		
Oscillator selection		
Filter selection		
Filter faders		
Envelope faders (ADSR)		
Working on DAWs		
Different MIDI source (controllers & host)		
Multiple instances		

Table 1: Testing checklist



Figure 4: SaucyWaves in Logic Pro X



Figure 5: SaucyWaves in Ableton 10

The testing results were generally positive and satisfactory as the plug-in loaded fine in both Logic Pro and Ableton and was able to record MIDI and playback the sound. The plug-in also reacted nicely when jumping to different parameters such as switching oscillators, changing the envelope or playing with the filter cut-off during playback. Most of the testing specifications were validated except for the Filter selection and faders. These exceptions will be discussed in the following 'Encountered Issues' section.

Encountered Issues

Audio programming requires some experience in software engineering in order to optimise the performance of the software be it an audio application or synthesiser plug-in. Knowledge in multithreading is crucial for digital signal processing as most audio applications are based on real-time computing (RTC) and require a lot of CPU resources when performing high data manipulations.

One of the main functions used by JUCE components for manipulating audio or midi data is

```
void renderNextBlock(AudioBuffer<float> &outputBuffer, int startSample, int numSamples)
```

This function is in the heart of the SynthVoice class and takes a fixed size buffer of samples and updates it before being outputted. The filter issues mentioned in the previous section are caused by the fact that there is a function call inside a function call, see overlaid code in yellow:

```
void renderNextBlock AudioBuffer<float> &outputBuffer, int startSample, int numSamples)
{
    for int sample = 0; sample < numSamples ; ++sample)
    {
        for int channel = 0; channel < outputBuffer.getNumChannels();++channel)
            outputBuffer.addSample(channel, startSample, setFilter() * 0.3f);
            tailOff *= 0.99;
        }
        ++startSample;
    }
}
```

This code is not optimised at all for DSP programming as it creates a lot of function calls when the CPU should be processing information quickly. The issue occurs when the user is playing with the cut-off frequency or the resonance fader and moves them too fast, which results in a massive crash of the whole sound system of the computer requiring the application to be closed out in order to solve this problem. This has to be fixed by either using the readily available JUCE DSP modules such as State Variable Filter, IIR or FIR filters, or by making a change in the design of the code.

Another issue previously mentioned is the band-pass filter. The selection of the filter in the menu section works perfectly fine, but when the user moves the fader, the same audio crash occurs. The Maximilian library creator added the following comment: *"This works a bit. Needs attention"* above the actual function implementation which further suggests that this aspect is still a work-in-progress in need of improvements and fixes.

Comparison & User Feedback

The synthesiser plug-in market is incredibly large and is full of very powerful and different synthesisers; there are several very famous synthesiser plug-ins that lead this market such as Native Instruments bass synth *Massive* or more recently Xfer *Serum*.

Saucy Waves is still a prototype itself so it certainly needs some programming changes and improvements as well as a few extra features in order to adequately compare to other products currently present on the market.

That said, in order to get some user feedback on the product in its current state, the plug-in has been sent to a group of producers in both VST and AU format. After taking some time to adequately test it out and familiarise with its full capabilities, the group was asked to complete a survey depicting their experience with the product and providing their subsequent feedback.

Even though the sample of users that were able to test the product and provide feedback was rather small, the answers were mostly positive and the overall feedback was good in general. Users loved the simplicity of the plug-in design and its intuitive usage as it only has a few controls. Some users also liked the sounds the synth produces. A help/instructions section was suggested as a potential improvement as well as an update to the overall GUI/look. From a commercial standpoint, users would generally buy this product for less than twenty pounds. The results can be found on this [link](#).

A detailed description of how the plug-in can be setup and utilised can be found in the Read Me of the project's GitHub page.

Improvements & Vision

As a first prototype there is a lot of optimisation to be added especially within the audio processing units. For instance, instead of using the Maximilian library's filters, it would be more useful and safe to use the JUCE DSP APIs as they are more stable. These APIs use specific DSP elements such as finite impulse response (FIR) and infinite impulse response (IIR) filters which are very common in digital filtering. The fact that JUCE is owned and developed by a known commercial company guarantees stability, customer service and frequent updates of the framework.

Another interesting feature to be developed would be the ability to support MIDI Polyphonic Expression or MPE. This would enable users owning ROLI products like Seaboard and BLOCKS to use the plug-in and achieve new expressive performances.



Figure 6: Serum Synth © Xfer

At the moment the Saucy Waves plug-in contains the most basic elements necessary to make sounds, but it is lacking in originality and would need to incorporate more features and other effects such as reverb or delay. It could also be improved by enabling two more oscillators and base all of them on wavetable synthesis so the user could have more choices or even import its own waves. In terms of user interface, the GUI was need to be properly designed using nice and appealing graphics in order to attract customers. The UI will need to be optimised for easy and intuitive usage as well.



Figure 7: ROLI Seaboard - © Roli

Some updates to the user interface have been added since the presentation. The following figures illustrate the evolution and the changes:

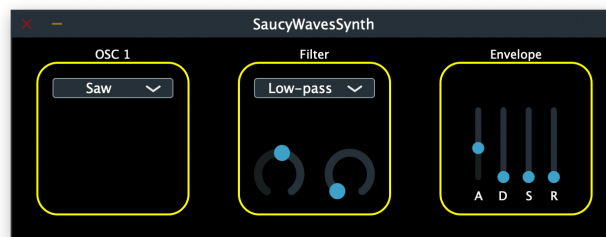


Figure 8 : First user interface design.

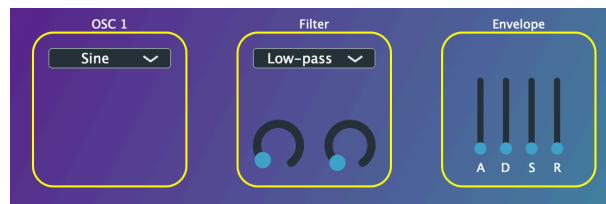


Figure 9 : Interface with gradient background.

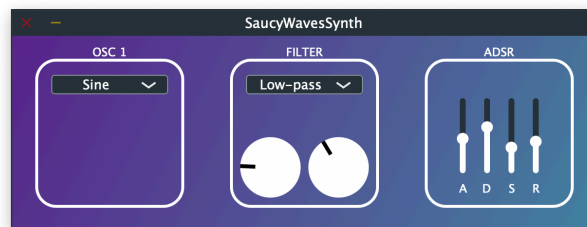


Figure 10 : Update to the filters and envelope looks.

Even though all these new features would add more value to the product, feedback from user testing and from the project presentation indicated that users were appreciative of the simplicity and look of the synth, as it presents a design that is not too overwhelming like other products on the market may appear to be. Once the product is stable, the vision would be commercialising it as an artist plug-in like Dutch Dj Nicky Romero did with his *Kickstart*. This strategy would get the fans to purchase the product.

Conclusion

This project has made it possible to develop the prototype of a potentially commercial synthesiser plug-in. Although the prototype does not work perfectly yet, it is a great start with room for improvement to eventually evolve into something marketable and useable. Furthermore, the experimentation, implementation and overall production of this product inspired many learning experiences such as familiarisation and newfound knowledge of the world of audio programming and MIDI protocol. Another significant learning outcome was the newfound knowledge of product design, particularly the understanding of how a given product may fit into a certain market and which targets would satisfy the commercial purpose. The importance of simplicity and user friendly interfaces has shown to be an important criteria as well from the user testing feedback.

The Creative Music Technology week for this module has helped to open people to the potential of music hacking and software programming whilst enjoying themselves.

References

1. Maximilian C++ Audio and Music DSP Library, Retrieved from GitHub <https://github.com/micknoise/Maximilian>
2. JUCE API Documentation, Retrieved from <https://docs.juce.com/master/index.html>
3. JUCE Projucer, Retrieved from <https://juce.com/>
4. VST / VST3 , Retrieved from https://en.wikipedia.org/wiki/Virtual_Studio_Technology
5. Apple Developer Documentation, Audio Unit AU, Retrieved from <https://developer.apple.com/library/archive/documentation/General/Conceptual/ExtensibilityPG/AudioUnit.html>
6. Aurélien Alexandre Merz. "Bluetooth Low Energy & Algorithm". Semester Work, University of Applied Sciences of Western Switzerland, Sion, May 2014.

LINKS

1. Survey results <https://www.surveymonkey.com/results/SM-X5BYS2DFV/>
2. Aurélien Merz Github project page: <https://github.com/aurelienmerz/SaucyWavesMidiSynth>

List of Figures

1 Saucy Waves plug-in v1.....	5
2 Projucer desktop app.....	6
3 Saucy Waves plug-in inside the host simulation.....	8
4 SaucyWaves in Logic Pro X.....	10
5 SaucyWaves in Ableton 10	10

6 Serum Synth © Xfer	12
7 ROLI Seaboard - © Roli	13
8 First user interface design	13
9 Interface with gradient background	13
10 Update to the filters and envelope looks	13

List of Tables

1 Testing checklist	10
---------------------------	----