Investigating the Viability of a Modular Synthesis Virtual Simulation in Python

UNIVERSITY OF
LINCOLN

Thomas Clare
CLA19704945

19704945@students.lincoln.ac.uk

School of Computer Science
College of Science
University of Lincoln

A report submitted in partial fulfilment of the
requirement for BSc(Hon) Computer Science

Supervisor: Dr. Wenting Duan

June 2022

Abstract

This project presents a software solution that utilises recent Python advancements to allow developers to build new and interesting modular synthesis modules using libraries that they are already comfortable and familiar with. Various assessments were performed on the software solution that showed that its audio quality, ease-of-use, and efficiency were appropriate for this task, and a compelling example module was built with PyGame to demonstrate this capability.

Table of Contents

List of Figures

List of Tables

1 Introduction

Musical synthesisers are much like other musical instruments. They create sound that, when played in the right way, creates music. Modular synthesisers take this idea further. A modular synthesiser is a musical synthesiser built from multiple discrete modules. Each module usually has a selection of inputs and outputs which are able to feed into and out of each other. Due to the variety of modules that exist, it is possible to create a musical instrument that arguably plays itself, in the way that the music will continue to play, even when the artist is no longer interacting with the instrument.

This is made possible with the idea of Control Voltage (CV). This type of signal acts as data and can be used to control other modules in the synthesiser. For example, an oscillator may produce a particular note. The artist, by tweaking a physical control, may be able to alter the pitch of the oscillator up and down. Consider another example where a different module is able to output a repeatable signal. That repeatable signal, being used as a CV signal, could be fed into the oscillator to control the pitch. This would be just like the original pitch control, but now the instrument plays a melody all by itself. The sum of these connections between modules and the resulting audio is called a “patch”.

Note that in order to bring about the added layer of automation and complexity, more hardware is needed. Sequencers, multiple oscillators, and audio effects are all common parts of modular synthesisers, and this extra hardware increases the financial cost for the artist. Non-musical hardware is also required to run these instruments such as power supply systems and the physical cables themselves, which adds even more upfront cost for artists to pay.

Modular synthesis software largely solves the problem of cost. With no physical product and quicker development timeframes, the product can reach the consumer at a considerably lower price. Modular synthesis software has been in the hands of consumers since at least the year 2000, with the launch of Reason v1 allowing artists to create patches using simulated modular components (Reason Studios, undated). VCV Rack was released in autumn of 2017, providing a C++ based open-source alternative to Reason (VCV, undated). Users can create their own custom modules, meaning any hardware-based module can be simulated and shared in the VCV Rack community library. Many modules have the same capabilities as widely used hardware modules but are available for free or for a fraction of the cost of their hardware alternatives. C++ is capable of multi-channel audio output rates, so it is an appropriate language choice for such a program.

This report discusses recent advancements in Python which would allow it to support such a platform. Exposing a musical audio engine to Python’s many mature libraries could allow previously unseen audio modules to come into existence. For example, data science tools, such as pandas, could use pre-existing or real-time data to manipulate musical components. This project aims to bridge the gap between Python and modular synthesis to create a platform where Python developers can create new and interesting ways to use existing libraries with audio and music.

2 Background and Literature Review

It is true that software simulations of modular synthesisers exist already and have been available to the general public for a long time. Reason Studios made this technology available to the public in 2000, and VCV Rack have made an open-source C++ implementation that has been available since 2017. However, being in C++ may be limiting how many programmers can interact with this system.

Python is the most commonly taught language amongst Australasian education institutions (Mason, 2017), and it would be appropriate to say that this may be the case worldwide. This means that a modular synthesiser system written in Python could potentially reach more programmers than one written in C++.

The study of Australasian institutions also showed that Python has been indicated to be relatively easy to learn (Mason, 2017), meaning those with a history only in music may be able to pick up the Python language more easily than C++ and become involved in the creation of custom software modules. This is a large benefit to a system using Python, because the musicians themselves are able to get directly involved and engineer modules to fill an exact role. This removes guesswork on the part of the software publisher and improves the appropriateness of the tools available on the platform.

However, creating a modular synthesis system using Python is difficult, as a limiting factor is speed. Audio data demands a bitrate that is rather high for live computation and Python simply does not possess the speed to produce these high bitrates when compared to C++. Zehra et al. (2020) compares the execution times of various algorithms in each language. Searching, sorting, inserting and deletion algorithms were assessed in this paper and, while it is true that these four algorithms can certainly not paint a complete picture, they help to give an indication of what real-world performance may look like.

The aforementioned paper also takes a look at the aspects of each language and identifies that Python would be an easier language for beginners to learn, but some arguments made are weak and unconvincing. For example, a point made against Python attempts to convince the reader that Python's simplicity has a negative aspect, as programmers may then have real difficulty to work with the "troublesome" existence of curly brackets and semicolons in other languages. Instead of arguing an effective point against the use of the Python language, this argument unwittingly gives criticism to languages that use braces and semicolons in syntax, as it is the existence of them in that language that makes that particular language more difficult to learn. It is not the responsibility of Python to make other languages easier to learn. The paper also makes the argument that Python is not a memory-efficient language twice, by framing a consequence of that fact as another point, separate from the fundamental issue. These issues come together to weaken the conclusion drawn by this paper.

Regardless, the observations of speed and memory-efficiency are sound, so it seems C++ would be more appropriate than Python to create real-time audio generation software. However, work has been done with Python to increase speed using the C language with one such project named Cython (Behnel, 2010). Cython is a Python library that enables C processes using the Python syntax, meaning Python can harness the speed of the C language while the programmer uses the more easily understood Python syntax. A benefit of using something like Cython, which Behnal's paper explains, is that only minor adjustments need to be made to the Python code to take advantage of the C processes. This means that, by targeting parts of the code that are both commonly run and inefficient, a lot of performance increase can be seen with very little work. The paper is informative and descriptive about how one might go about implementing Cython into their Python project. It also highlights some clear limitations of Cython and how they may be mitigated, which overall forms a solid literary work on the project.

Alternatively, Python's speed can be improved by using the C API, which is the preferred method of a digital signal processing (DSP) library called Pyo (Belanger, 2016). In Pyo, all control structures and audio chains are written in Python, and all the DSP code, which is more mathematically intensive, is written in C. These two

parts are connected via Python's C API, where Python can call predefined C functions that can generate the audio data in a timely manner. Pyo contains a number of waveform generators prebuilt, such as sine waves, supersaws, and triangle waves. Pyo is a good library to build upon for this project because having prebuilt audio engines will allow more development time to be spent working on the data flow and usability of the program.

The control structure will rely heavily on a between-class data flow. The musical pipeline usually necessitates some rhythmic flow of data towards a tone generator, which then travels through audio effects to produce the final musical sound. For example, a square wave generator may be producing a beats per minute (BPM) signal, where every time the wave reaches an amplitude of 1, a new beat is indicated. This BPM signal could be fed into a sequencer that triggers the next value in some sequence every time the input reaches 1. This output value could be used to control the pitch of a sine wave generator, producing a simple musical loop. A programmatic implementation of this may be difficult to achieve in python. The connection between the BPM signal and sequencer is a good example of the problem. The sequencer needs to check the incoming signal every time it changes, to see if it is equal to 1. A simple solution is to put an 'if' statement inside a 'while' statement that loops forever, however this is a very inefficient method and will result in audio artefacts. This will need to be navigated as the project continues.

Python has many mature libraries that can bring new abilities to software modules. Machine learning libraries can be used to improvise melodies, shown in a study of how machine learning can improve algorithmic music (Chan, 2006). As a stretch target for this project, responsive visualisations will be created with a library called PyGame, where musical parameters could be turned into input and the music itself manipulates shapes and visual effects. Usually for music visualisations, feature extraction proves to be what the majority of the work surrounds (Graf, 2021), but since the music generation is part of the code, those features can be referenced directly. There are numerous guides to PyGame that will be relied upon throughout development (Sweigart, 2012).

3 Methodology

3.1 Project management

For this project, the Kanban methodology was used. Careful thought was given to this decision, as it has been shown that choosing the correct Project Management Methodology (PMM) and tailoring the PMM to the project are two key factors in successful projects (Joslin and Müller, 2015, 1388). Here, the PMM choice and any variations will be discussed with specific relation to the nature of the project.

It is important to keep a number of factors in mind when choosing a PMM. Cockburn (2000) provides an elegant breakdown of four different principles that can help the project leader to understand, differentiate, and select appropriate methodologies. These principles are mostly appropriate to projects requiring teams, but some are appropriate for individuals undertaking projects on their own. The applicable principles will be looked at in turn and applied to the project at hand, justifying the use of the Kanban methodology and specifying which adjustments are appropriate for this project.

"A larger team needs a larger methodology." It would be easy to assume that, since this project is being carried out by a single team member, the methodology should be equally as minimal. However, Cockburn makes the astute observation that PMMs should not grow with the number of people in the team, rather they should grow with the number of roles in the team. It would be fair to say that PMMs assume that communication between roles is a problem to be solved but, in this project, the individual team member plays all roles. This brings a more nuanced context when considering this principle. Any advantages offered by a PMM that relates specifically to communication between team roles should be given deeper thought and often cannot be taken at face value. Considering this, Kanban is lighter on communication when compared to other methodologies such as Scrum (Orlov et al., 2021) which is appropriate for this project since no team meetings would need to take place. This lack of necessity for team meetings does not mean their use should be wholly ignored, however. Team meetings provide ample opportunity for reflection and task prioritisation.

To avoid losing focus throughout the project, it is important to keep reflecting on the initial intent. Defining the critical path of the project before beginning can provide a point of focus for the project. There are many parts of this project that necessarily rely on being delivered in a particular order. For example, the software capabilities will need to be modelled on the software alternatives in order to deliver a comparable solution. The critical path of the project identifies which features the project requires, which features the project can succeed without, and how long each task would take to complete. Using a similar idea, it is possible to define a core task path for the project with respect to task dependencies and ensure core tasks are completed on time and in order. This will also result in task priorities, which will be labelled on each task item.

"A more critical system – one whose undetected defects will produce more damage – needs more publicly visible correctness (greater density) in its construction." This principle outlines how the accountability of the project affects the PMM chosen. Should the software fail, there are different levels of disruption it could cause, depending on the nature of the software. Should this project fail, it could cause some monetary damage to the user. However, that is only if the user is using the software in a commercial setting without having tested it before. This project does not aim to make commercially viable software. The aim is of an academic nature - attempting to allow more programmers and musicians to become involved with creating custom modular synthesis modules. Should the software fail, it would cause a loss of comfort and convenience for the users until it is fixed. Such a situation is less serious than irreplaceable financial cost and far less serious than a loss of life, the latter of which may be a consequence of unreliable software in aerospace and engineering applications. Therefore, the required density of the PMM can be comparatively low. This is another way in which Kanban is appropriate.

"A relatively small increase in methodology size or density adds a relatively large amount to the project cost." This third and final applicable principle states the importance of minimising the density of the selected PMM where possible. There may be many PMMs that would be appropriate for this project, but where density can be reduced, it should be. When implementing Kanban in this project, the toolset was carefully selected to abide by this rule, which is further discussed in the section titled *Toolsets and Machine Environments*, which helped to further reduce the time cost of this project.

3.2 Software Development

This project is of an exploratory nature. This means that it will not be the breadth of features which would bring success to this project but rather a central element that is well suited to fix the problem. For this reason, the project is likely to result in a small amount of solid code. This is opposed to a large amount of "good enough" code that may solve more problems or cover more use cases.

Since code improvements will likely often result in re-written code, it is important for each work item to hold accountability when it breaks a previously existing feature. This means that before calling a feature "completed", the whole codebase must undergo a full test. If any previous feature is found not to work after implementing some new feature, then correcting that error would become the responsibility of that task. This idea is applicable to many methodologies and would not suggest a particular methodology on its own but is still an important aspect to consider in the context of this project.

For a more direct suggestion, Cockburn's third principle can be revisited. In the absence of more intense requirements in this project, the least dense appropriate methodology can justifiably be selected. This line of reasoning may suggest the Kanban and Scrum methodologies as they are both fairly lightweight methodologies. It makes sense to refer back to which PMM has been selected, as it is possible to use the same methodology for project management and software development. Reusing the same tool between project phases reduces the overall complexity of planning and tracking the project, meaning that using the Kanban methodology during software development is a justified choice.

Applying the work item accountability tweak to the software development phase of the Kanban method ensures that this methodology is both simple *and* thorough for this project's aim and scope.

3.3 Toolsets and Machine Environments

The tool used to keep track of the PMM must be carefully selected. If it does not have specific features that are required by the PMM, then the project's success will ultimately be affected in some way. It is for this reason that a careful consideration must take place. In order to justify whether a particular tool is a good fit for the project or not, some specific requirements must first be decided.

As this is an exploratory project, there will be many lessons learned about the base libraries, environment, and objectives as the project continues. There will also be caveats to new features that must be kept in mind moving forward. Therefore, it is important that the tool allows work items to have notes attached to them. By making sure that lessons are only learned once, there will be less repeated mistakes and the time efficiency for the project will be improved.

The workflow will need to be reflected upon when evaluating the project methodology success. It is important to know when tasks were completed so that those completion times can be compared with the project plan. This provides the opportunity at the end of the project to look retrospectively at each task and can help to identify any problem areas.

The idea of work item accountability has been discussed earlier in this report. Using subtasks would be a very appropriate method of keeping track of any issues or consequences of implementing a new feature. It would serve a similar purpose to a notes section but would hold information that needed to be specifically acted on.

Having a portable tool would be very beneficial, as talking to participants in a place that is comfortable for them will more than likely mean working away from the usual working place. Unfortunately, this means that a physical Kanban board may not be appropriate because it can be difficult to move. In the case of Kanban software, it means that it must be portable between systems, as another metric for this project will need to be measured by working on multiple computers that are physically situated in different places. These two requirements will be discussed and justified in the following section titled *Research Methods*.

Tagging work items can help keep track of specific areas of the project. This is not as important as the other requirements as the critical path partially fulfils this role.

Some tools were researched and shortlisted for their reasonable performance against the aforementioned important aspects and are compared in the matrix below. There are two additional columns which provide context around the tools themselves. The first is titled *Kanban formatting* and refers to how the tool works visually compared to the standard Kanban layout. The second is the *Cost* column and describes the financial commitment involved. Both of these additional columns are informational in nature, as opposed to the other columns that each describe an important functionality the tool is being assessed for.

Tool	Extended notes	Board history	Sub-task	Multi-device Sync	Subject tags	Kanban formatting	Cost
Trello	As comments						Free option
Toggl			Available in paid				Free option
Google Docs		Via version history			No linking	In list format	Free
Zoho	As comments						Free option

Table 1: Project management methodology comparison

When reading the above matrix, it is important to understand that the tool *Google Docs* does not offer these features directly. It allows the user to construct these features on their own in the same way that a physical Kanban board might. For example, sub-tasks can be implemented by indenting a task under a parent task. The relationship is represented in a purely visual manner, whereas other specialised software will have a data point storing that connection.

As the matrix shows, it seems that *Google Docs* is the most appropriate tool. Using more specialised software may mean the PMM may have to go without particular features, or additional tools may have to be used to cover the requirements, and the open assertion-less nature of a text document offers the most flexibility. This is due to the unique requirements of the project, and should a version history not be needed or if there were more team members involved on the project (and therefore more communication be required), the outcome would likely have been different. This process has highlighted the importance of individually assessing the needs of the project and not blindly selecting a tool due to convention or familiarity.

When it comes to selecting a machine environment, a similar approach is needed. This project particularly works with audio, so it would be beneficial for the Integrated Development Environment (IDE) to be able to open audio files. This would prevent having to open a new program when previewing audio samples, saving time throughout the project.

This project will make use of Git for file version management. This has been selected because of its many rigid features and its submodules feature, which will help to import the required libraries to make this project work as intended. Virtual environments will also be used to manage packages and ensure a suitable environment across computers. With these two factors considered, it would be useful for the IDE to have an integrated terminal. This would help to more easily manage the additional tools being used.

If an IDE can integrate with the Python interpreter, it allows the developer to debug the code as it is being run. Bug fixing consumes over 50% of development and maintenance costs (Collofello and Woodfield, 1989), so it is important to reduce this cost where possible. This can greatly decrease time spent troubleshooting and is therefore an important requirement.

In the matrix below, the *Cost* column is purely informational and does not affect the IDE selection.

Tool	Interpreter with Virtual Environments	Built-in terminal	Audio file compatibility	Cost
Jetbrains Pycharm				£14.90 p/m
Atom	With setup	Through extensions	Through extensions	Free
IDLE	Only on startup			Free
VS Code			Through extensions	Free

Table 2: Integrated Development Environment comparison

Both Atom and VS Code would successfully be able to satisfy requirements for this project. However, VS Code would require less setup, as Atom would require more extensions to satisfy the same project requirements. For these reasons, VS Code has been selected for this project as it satisfies all project requirements and needs minimal setup.

3.4 Research Methods

The assessment of the project's success will directly depend on what the aims of the project are. This project aims to bring Python and modular synthesis together to create a platform where Python developers can create new and interesting ways to use existing libraries with audio and music. There are a number of components to this aim. Each component will be identified and discussed in turn.

Firstly, the project must use modular synthesis ideas to create a musical platform. To measure how well the project has achieved this, a test must be developed that can assess how musically capable the platform is when compared to other modular synthesis software. An audio fingerprinting algorithm would provide appropriate results (Kekre et al., 2013) in a quantitative manner. However, the results should be from a listener's perspective, whereas audio fingerprinting methods are a purely algorithmic approach. Some other human perception-centred approach would be more appropriate to measure this aim.

To do this, the musicality of the software can be measured by producing two identical tracks - one with the C++ based alternative, VCV Rack, and the other with the project software. The participant can be shown either one of these tracks without knowing which track they have been shown. Then the participant can listen to both tracks and will attempt to specify which track they heard initially. The measurements will be of a nominal type. This may happen multiple times with different pairs of tracks, and the success rate of each pair of tracks can be presented to show how often they were confused with each other by participants. The source of the initial track will be randomised for every participant in each round of tests.

Multiple pairs of tracks should be created for this test in order to get a wide range of base features. If some are shown to sound very similar and others are shown to sound different, it could indicate that some specific parts of the software have a notably inaccurate output when compared to VCV Rack. This type of test would show up timing errors and audio artefacts, which are both important for clean audio output.

This project aims to be able to reach more programmers than C++ based systems. Ease-of-use and system requirements are both barriers for programmers to overcome. Therefore, these will be tested in an appropriate manner.

Module developers rely on documentation and tutorials in order to learn how to craft their modules. This can be demonstrated by the fact that VCV have an entire portion of their website devoted to plugin development. A link is available in the appendix. Documentation will be produced for this project to serve the same purpose of educating developers about the project software. Those who have experience in both C++ and Python can be asked for their opinions about creating a custom module with this project's platform, the Pyo platform, and the VCV Rack platform. This will provide the perspective of the eventual users of the project software and can help to identify any shortcomings of the solution. Here, it is appropriate to collect both quantitative and qualitative data. Quantitative results will provide more objective and presentable statistics surrounding the ease-of-use of the platform. Collecting qualitative data will capture the reasons for the results, which will help to gain more insight on how the project has been received.

Performance measurement would be most appropriate as a quantitative measurement, as it is an objective observation about the operation of the program. A ratio measurement would take place to measure how much of the CPU time is being taken up by the program as a percentage when generating the audio tracks. As each track has a different density of instruments, this would show how performance grows and complexity increases. There would need to be multiple measurements to establish a fair average result. These averaged results would then be displayed in a bar chart.

Finally, the ability to use the project software to create new modules with pre-existing libraries was a positive consequence of the project. The stretch target for this project was to demonstrate this ability by creating an audiovisual system which used the base components of the audio output to create a visual show suitable for venues or art installations. This stretch target was completed but, as it functions as a proof of concept, it will not be assessed in this report.

4 Design, Development, and Evaluation

4.1 Software Development

The following sections refer to the main project software solution. Requirements, design, build, and testing will be discussed while considering operation and project aims throughout. Afterwards, the audio visualisation addon will be briefly discussed. This component is a stretch target and is not the main focus of the project.

4.1.1 Requirements

The aims of the project will largely dictate the requirements of the software. This project firstly needed to match a core functionality offered by VCV Rack. The best way to decide which functionalities are core functionalities was to create simple tracks with VCV Rack and then observe the modules used to make the sound. The project software would need at least these modules in order to offer this same functionality.

Ease-of-use was another aim for the project, which was mainly achieved through selecting the correct base libraries in the research phase of the project. If a non-user-friendly library was selected, it would be much more difficult to make the project software user friendly. However, if a simple base library was selected at first, it would be easier to maintain ease-of-use.

Finally, the software must allow developers the ability to add code that interacts with the musical composition. This is a requirement that needs to be solved in the design stage of the project, as it necessarily depends on the structure of the software.

4.1.2 Design and Build

No mathematically intensive software design needed to be done as the base library Pyo would handle audio generation. However, thought would need to be given to how the user would ideally interact with the software. As tool familiarity has been shown to increase user performance (Tomasi et al., 2018), the user would largely benefit from a design similar in logical structure from VCV Rack. That is to say that code classes would benefit from being imported and used as would modules in modular synthesis.

Looking at the target tracks that were made with VCV Rack, the project software would need some version of each of the following:

- Master Clock
- Sequencer
- Drum sample player
- Sine wave oscillator
- Reverb
- Mixer
- Audio output

Pyo can be relied upon for the reverb and audio output, which will remove some workload for the project.
In an effort to reduce the complexity for developers new to modular synthesis, however, the decision was taken to merge some modules for this project. Firstly, the sequencer was integrated into the audio generation components, creating a drum sequencer class and a sine wave sequencer class. Secondly, volume levels were made a property of each of these classes, removing the need for a separate mixer class. These two changes would both reduce the complexity and workload of the project, while still offering the same functionality and familiarity for users of all experience levels.

This means the project will consist of three main classes. Figure 1 shows each class and their summarised role.

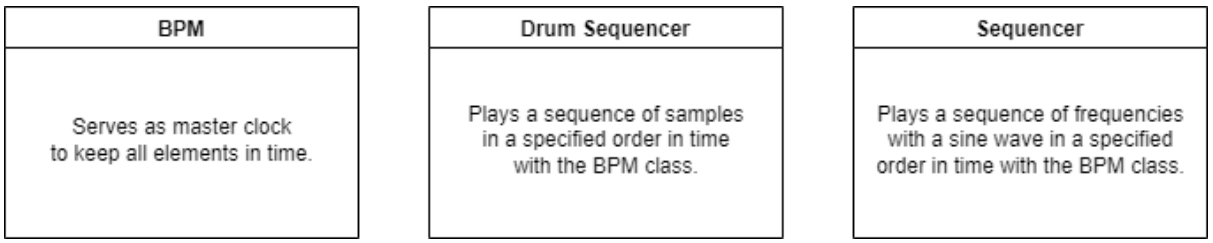

Figure 1: Summarised class diagram

To produce a track, there may be varying amounts of instantiations for each class. For example, the third track produced by VCV Rack for this project's assessment contains three different drum sounds and one sine wave synthesiser sound. Figure 2 shows how that can be represented with the classes designed for this project.

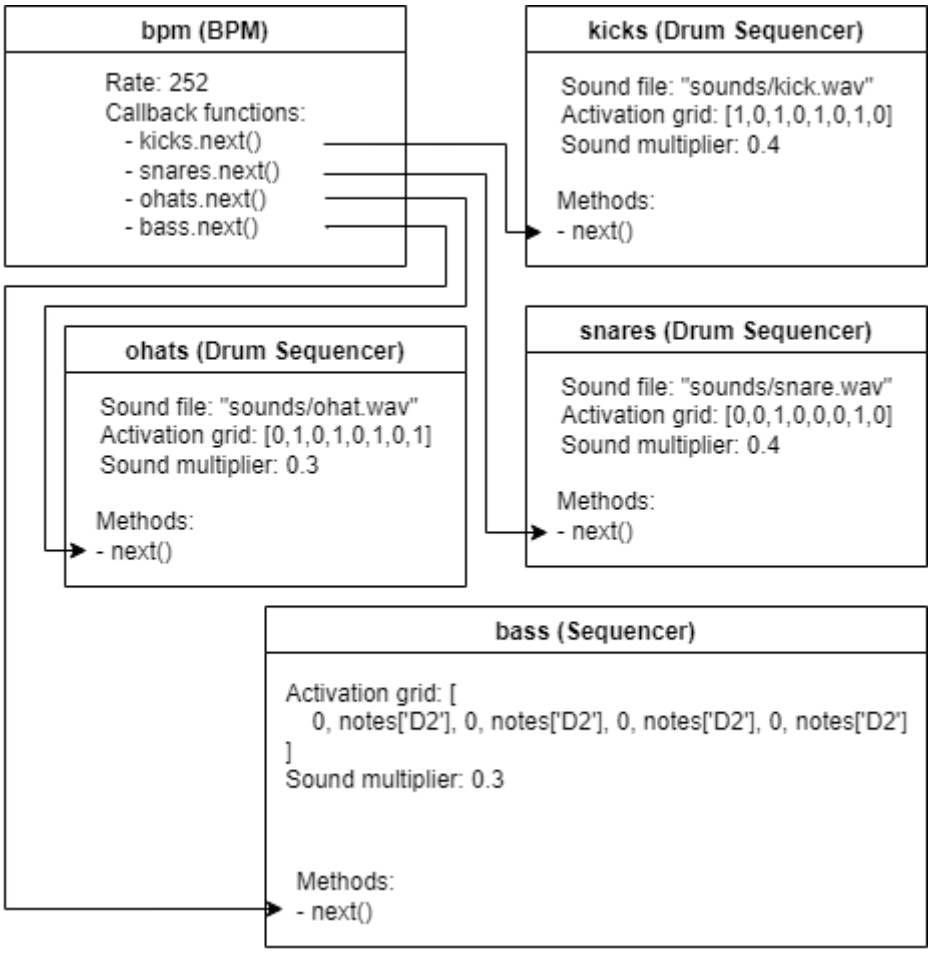

Figure 2: Runtime composition of classes

The single instantiation of the BPM class will create a constant tick, the speed of which will be defined by the '*rate'* variable. The interval can be calculated by dividing 60 the '*rate'*. At the end of each interval, each callback function will be called. The list of callback functions supplied are the '*next()'* functions of the sequencer classes. The sequencers '*next ()'* functions will make the sequencer output the next value in their '*activation grid'*. This '*activation grid'* can be filled with binary gates or with frequency values, which is reflective of their hardware counterparts. If the next value in the grid is 0, nothing is output on this beat.

The ticking of the BPM class can be built simply as the '*go()'* function seen in figure 3. Each callback function is called and the loop pauses for the duration of the interval. This function needs to be called on a new thread so that the rest of the script can produce the audio components, which can be done using Python's *threading* functionality.

```
30
31      def __init__(self, rate, func_next):
32          self._rate = rate
33          self._duration = 60 / self._rate
34          self._func_next = func_next # this must be given as a list
35
36          self.heartrate = threading.Thread(name="heartbeat", target=self.go, daemon=True)
37          self.heartrate.start()
38
39
40      def go(self):
41
42          while(True):
43              for func_next in self._func_next:
44                  func_next() # call callback
45              sleep(self._duration) # pause for tick interval
```

Figure 3: __init__() and go() functions of the BPM class

Figure 4 shows how the patch is created with the project code. Note how each module is created, described, and started all within a single line. It can be seen how the callback functions are supplied to the BPM class. This is a possible entry point for other modules to synchronise with the project.

```
## simple offbeat with hihat and bass
s = Server().boot()

kicks = SoundSequencer("sounds/kick.wav", [1,0,1,0,1,0,1,0], 0.4)
snares = SoundSequencer("sounds/snare.wav", [0,0,1,0,0,0,1,0], 0.4)
ohats = SoundSequencer("sounds/ohat.wav", [0,1,0,1,0,1,0,1], 0.3)

bass = Sequencer2([0,notes['D2'],0,notes['D2'],0,notes['D2'],0,notes['D2']], mul=0.3)
bass_mix = bass.mix(2).out()

bpm = BPM(252, [kicks.next, snares.next, ohats.next, bass.next])

s.gui(locals())
```

Figure 4: Example script

The audiovisual extension created as a stretch target for this project uses this entry point. A PyGame environment is created on a different thread, and a function that randomises the position of a shape is added to the BPM's callback functions. This results in the shape changing its *x* and *y* coordinates every time the BPM counter ticks. A link to this code can be found in the appendix.

4.1.3 Testing

As discussed in the *Methodology* section, all code needs to be tested before a single feature is marked as complete. In order to test all features, a script was created that helps the developer to observe all behaviours. A feature checklist was used to ensure nothing was missed. Figure 5 shows a snapshot of the checklist partway through development.

- Ensure Pyo dependencies run
- Audio output heard
- Audio is synced across classes
- Drum sequencer activation grid and audio is consistent with each other
- Sine sequencer activation grid and audio frequencies are consistent with each other

Figure 5 - Testing checklist partway through development

This checklist later went on to include audio panning, docstring syntax, and compatibility with Pyo's reverb module.

4.2 Audio Visualisation

This addon is built with PyGame and demonstrates how a Python developer might integrate an addon into this project or use this project in an existing project. A new window with a black background is created, and the environment is set up. A game loop that will draw all sprites to the screen and update the screen is defined inside a function. This game loop function is started on a new thread to ensure the rest of the script that contains core project code is executed. Three drum sequencers, a sine wave sequencer, and a BPM object are created. A sprite, part of the PyGame-related code, has a function called '*newPos'*, which randomises the position of the sprite appearing as a white square on the game screen. This means that with every tick of the BPM class, the position of the square changes in synchronisation with the music.

This is a very basic example intended to demonstrate what is now possible because of this project. This example could be taken much further. For example, the sprite could count the number of times the function has been called. With a modulo operator, it would be possible to update the position every *x* number of times, perhaps synchronising with a particular instrument in the track. Code from the sequencer class could be reused in the sprite so that the PyGame object has its own activation grid, acting as a visual instrument of its own. This addon was created to be intentionally simple so as to not overly-confuse those learning from it, while still providing a clear and concise example of how integration might be achieved.

4.3 Research component

This project has three ways in which its success is measured. Since each component varies in nature, each will be discussed in turn. Every participant that took part received an appropriate participant information sheet that gave informed consent for the specific study they took part in. This information sheet outlined the risks and consequences of taking part, if any. The information sheet also specified how they could raise concerns to appropriate authorities should they need to and explained that no personal information will be gathered during the study.

4.3.1 Audio Perception

4.3.1.1 Audio Perception Participants and Hypothesis

Participants should be of sufficient hearing ability. No specific skills would be required as the study aims to understand the software capabilities from the perception of the average music listener. Specifically, the hypothesis is as follows: *Simple audio outputs of VCV Rack and this project software are not meaningfully differentiable by the average listener.* To assess this, nominal data should be gathered around the perceptions of listeners to be able to calculate as a percentage how often outputs of the two pieces of software are confused for each other.

4.3.1.2 Audio Perception Setup and Process

To set up this study, multiple pairs of compositionally identical tracks must be created and exported into an audio file. The volume between each track version must remain identical, and no exportation artefacts can be present. This ensures that any differences in audio come only from the software that generated them and are not a result of unrelated preparation methods.

During each round of tests, the participant will need to hear a specific recording before any others. This may alter the participants perception of the subsequent recordings and impact the results of the study. This cannot be prevented, but it can be accounted for in the results. By increasing the variance of the tracks participants hear first, the results should remain reflective of the true result. To introduce this variance, a key was created that effectively spreads the conditions for the number of participants. This key can be seen in figure 6. Track 1, 2, and 3 were named “Simple Patch”, “Simple with offbeat hihat”, and “Simple with offbeat hihat and bass” respectively.

Par. No.	Simple patch	Simple with offbeat hihat	Simple with offbeat hihat and bass
1	simple_patch_vcv.wav	simple_w_offhihat_proj.wav	simple_w_offhi_bass_proj.wav
2	simple_patch_vcv.wav	simple_w_offhihat_proj.wav	simple_w_offhi_bass_proj.wav
3	simple_patch_vcv.wav	simple_w_offhihat_proj.wav	simple_w_offhi_bass_vcv.wav
4	simple_patch_vcv.wav	simple_w_offhihat_vcv.wav	simple_w_offhi_bass_proj.wav
5	simple_patch_vcv.wav	simple_w_offhihat_vcv.wav	simple_w_offhi_bass_vcv.wav
6	simple_patch_vcv.wav	simple_w_offhihat_vcv.wav	simple_w_offhi_bass_vcv.wav
7	simple_patch_proj.wav	simple_w_offhihat_proj.wav	simple_w_offhi_bass_proj.wav
8	simple_patch_proj.wav	simple_w_offhihat_proj.wav	simple_w_offhi_bass_proj.wav
9	simple_patch_proj.wav	simple_w_offhihat_proj.wav	simple_w_offhi_bass_vcv.wav
10	simple_patch_proj.wav	simple_w_offhihat_vcv.wav	simple_w_offhi_bass_proj.wav
11	simple_patch_proj.wav	simple_w_offhihat_vcv.wav	simple_w_offhi_bass_vcv.wav
12	simple_patch_proj.wav	simple_w_offhihat_vcv.wav	simple_w_offhi_bass_vcv.wav

Figure 6: Variance key used to introduce spread amongst starting conditions

The following process is followed for each pair of tracks that form part of the study:

- Notify the participant that the reference track will now be played.
- Play file specified in variance key.
- Notify the participant that track A will be played first and track B will be played after track A has finished.
- Play appropriate project and VCV audio files in a random order, keeping note of which was played first as track A and which was played second as track B.
- Ask the participant to specify which track (A or B) is the reference track.
- Record which track the participant perceived as the reference track and if this is correct or incorrect.

When each pair of tracks has been played and assessed, conclude the study with the participant.

4.3.1.3 Audio Perception Results and Analysis

Participant	Track 1	Track 2	Track 3
1			
2			
3			

4			
5			
6			
7			
8			
9			
10			
11			
12			

Table 3: All audio perception results

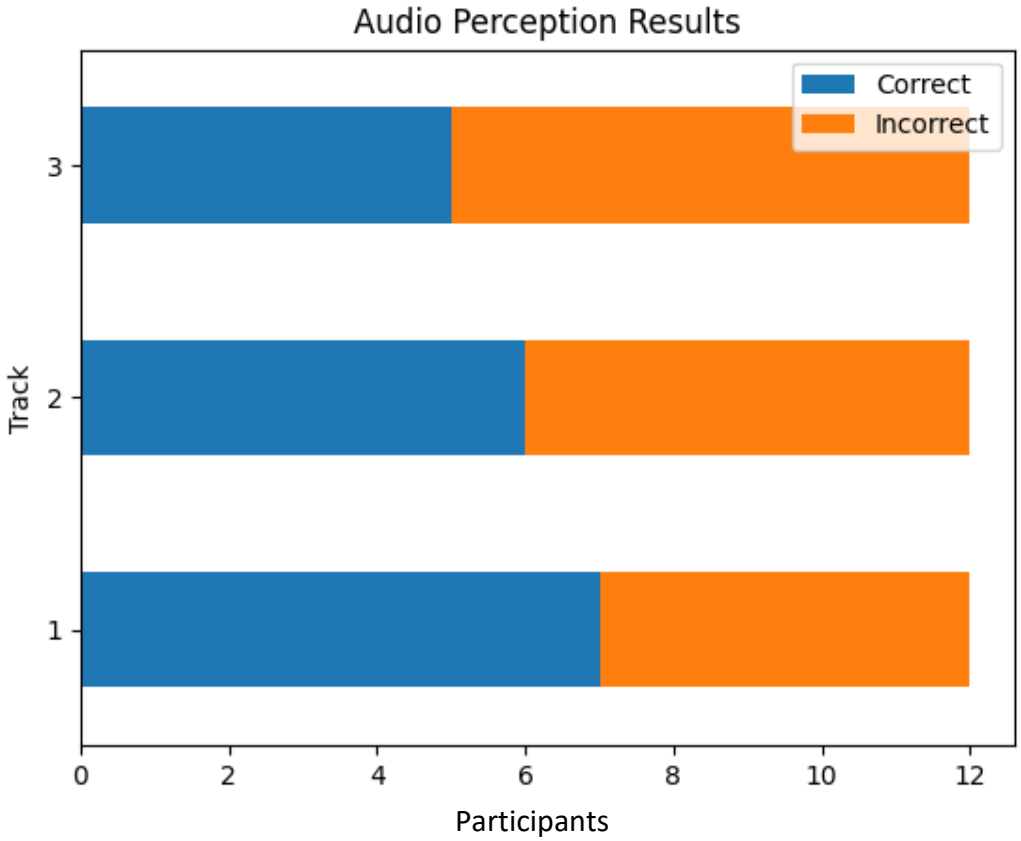

Figure 7: Audio perception results as a stacked bar chart

If each pair of tracks is extremely difficult for the participants to tell apart, a success rate of around 50% is expected. This is because there is a random chance that each guess is correct and there are two options. If three versions of each track had been available to choose and were difficult to tell apart, a success rate of around 33.3% would be expected. With this idea in mind, observe table 4.

Track	Correct %	Incorrect %
1	58.33	41.66
2	50	50
3	41.66	58.33

Table 4: Correct and incorrect perceptions as a percentage of total participants

The results for track 2 clearly fits this pattern. It could be argued that the results for track 1 and 3 also fit this pattern, but more participants would be needed in order to confirm this. However, these results have shown the project solution to, at the very least, nearly match the audio quality of VCV Rack when creating short and simple audio tracks.

4.3.2 Usability Assessment

4.3.2.1 Usability Assessment Participants and Hypothesis

Participants should have some experience with both Python and C++ in order to effectively relate to their experience and skills to give meaningful feedback. The hypothesis is as follows: *Developers find it easier to use this project software than VCV's software.* This hypothesis will help to assess multiple important parts of software usability - namely documentation readability, documentation comprehensiveness, and code understandability. Each element will be assessed by recording quantitative ordinal data of how much each participant agrees with a certain statement via a likert scale, with an opportunity for each participant to provide qualitative data about their reasoning.

Usability Assessment Setup and Process

A questionnaire was given to each participant. Each questionnaire contains three links. One to the documentation for this project, another to the documentation for the underlying library Pyo, and a final link to VCV's Plugin Development tutorial. Participants can engage with the questionnaire through a 5-point Likert scale and a text box where 1 was "disagree" and 5 was "agree". There are four sections of the questionnaire, each with a set of similar statements that the participant is invited to share their thoughts on.

Section one has the three following statements followed by a text box titled "Please share your reasoning.":

- The code from documentation 1 is easy to understand.
- The code from documentation 2 is easy to understand.
- The code from documentation 3 is easy to understand.

Section two is structured identically with the following statements:

- I would feel confident using the project 1 with only the documentation to support me.
- I would feel confident using the project 2 with only the documentation to support me.
- I would feel confident using the project 3 with only the documentation to support me.

Section three is also structured identically with the following statements:

- I would feel comfortable adapting the code from project 1 to further suit my needs.
- I would feel comfortable adapting the code from project 2 to further suit my needs.
- I would feel comfortable adapting the code from project 3 to further suit my needs.

Section four provides a link to a sequencer implementation recommended by the Pyo documentation and has a single statement:

- The Sequencer class in project 1 is easier to use than the sequencer script shown in documentation 2.

4.3.2.2 Usability Assessment Results and Analysis

The collected data was combined into table 5. The cells have been left blank in place of zero so the data is more easily read.

Section	Statement	Total Likert Occurrences				
		1	2	3	4	5
1	1			1	3	3
	2			1	5	1
	3		3	3	1	
		1	2	3	4	5
2	1				2	5
	2			2	4	1
	3		2	3	2	
		1	2	3	4	5
3	1			1	3	3
	2		1	3	3	
	3			4	3	
		1	2	3	4	5
4	1				4	3

Table 5: Results of the usability assessment

The mean average has been calculated for each statement and the results are shown in table 6.

Section	Statement	Average Likert value
1	1	4.3
	2	4.0
	3	2.7

2	1	4.7
	2	3.9
	3	3
3	1	4.3
	2	3.3
	3	3.4
4	1	4.4

Table 6: Averaged results of the usability assessment

The results from section 1 indicate that documentation from project 1 consistently seemed considerably easier to understand than project 3, but only marginally easier to understand than project 2. To find out why, data from the text boxes can be assessed. Clarity is mentioned by 4 of the 7 participants as a strength of project 1s documentation and thorough examples is mentioned by 3 participants. Project 2s documentation is praised for its excellent code comments by 4 participants, but 2 participants mention that vague variable names in the code examples may cause confusion. Presentation is noted as a weakness of project 3's documentation by 5 of the 7 participants with inconsistent naming conventions also being named by 2.

Section 2 measures the comprehensiveness and readability of the documentation. All three projects seem evenly spaced out in this regard. 3 of the 7 participants have said that project 1s documentation is more useful because it offers information beyond the docstrings. For project 2s documentation 2 participants said that it is too linear and fails to show the flexibility of the classes, and project 3 is said to have not enough examples by 4 of 7 participants. Overall, there is less information offered up by participants in this section than in other sections.

Section 3 explores how comfortable the developers would be adapting the code from each project with only the documentation to support them. Similar statements are echoed about project 1 and 2 by participants, but project 3 performs better here than it has done before. 5 out of 7 participants have said that project 3s documentation is comprehensive. Understanding this statement in conjunction with earlier statements about the project being hard to understand provides a clearer picture of the problem. It seems that project 3 is simply too large to effectively learn about just through documentation and should lean further on providing tutorials and examples.

These results show promise for the success of the project. The comparatively smaller size of this project aids the documentation to help unfamiliar developers become comfortable with the software. The documentation provides useful examples with clarity where alternatives struggle to communicate ideas effectively. This project aim has been achieved.

4.3.3 Performance Assessment

4.3.3.1 Performance Assessment Hypothesis

The speed test hypothesis is as follows: *The project solution performs similarly or more efficiently than VCV Rack.* This hypothesis is based on the project aim of lowering the barrier of entry for developers. The biggest alternative software to this project is the VCV Rack software. Therefore, if this project performs similarly or better than VCV Rack, then it can be said that the barrier to entry has been lowered in terms of system requirements.

4.3.3.2 Performance Assessment Setup and Process

To assess this hypothesis, the reference track generation files will be run on a computer in turn. Using Windows Performance Monitor, the process's % of CPU time is tracked. Performance Monitor displays an average figure which is highlighted in figure 8.

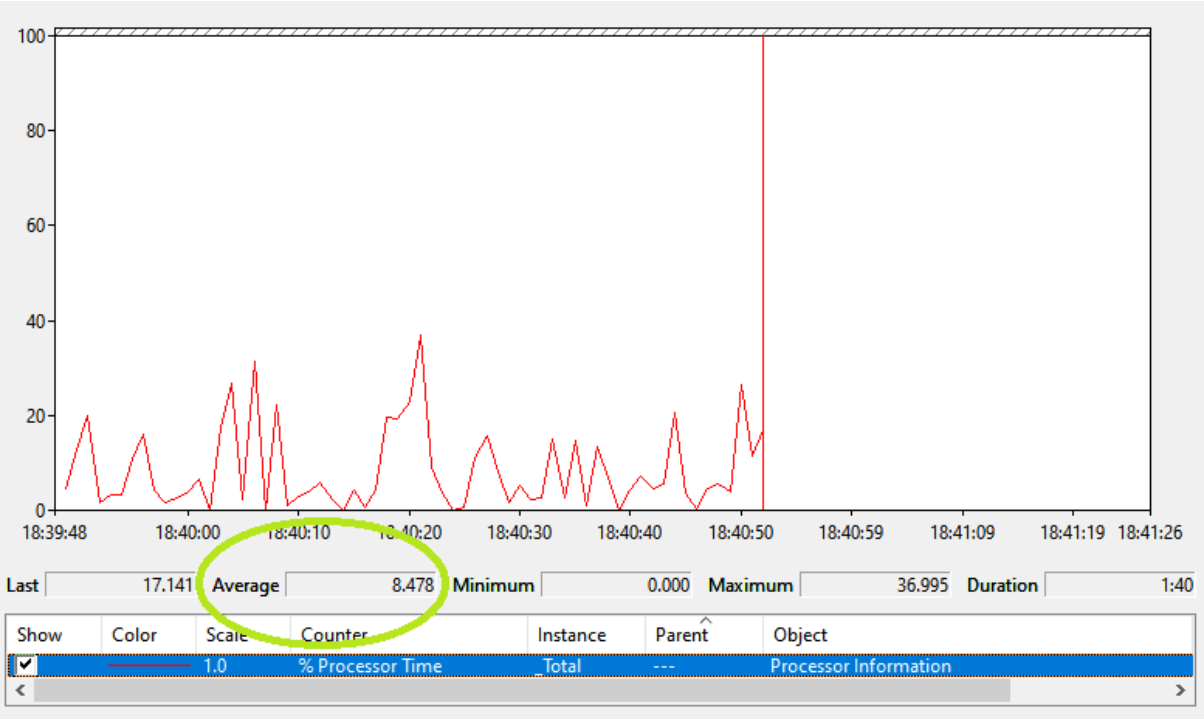

Figure 8: Performance monitor with Average measurement circled in green

Each track was run until the *Average* measurement stabilised, at which point it was recorded. This measurement was taken three times and an average was calculated to account for different running conditions.

4.3.3.3 Performance Assessment Results and Analysis

The results are shown in Table 7 and presented in a chart in figure 9.

Track	Attempt		% avg CPU time			Track Average (3 decimal places)	
			Project	VCV Rack		Project	VCV Rack
1	1		1.433	17.010		1.523	17.502

	2		1.586	18.521			
	3		1.550	16.974			
2	1		1.595	19.114		1.570	20.156
	2		1.563	21.739			
	3		1.542	19.614			
3	1		1.909	34.081		1.866	33.828
	2		1.869	32.782			
	3		1.821	34.620			

Table 7: Performance assessment results

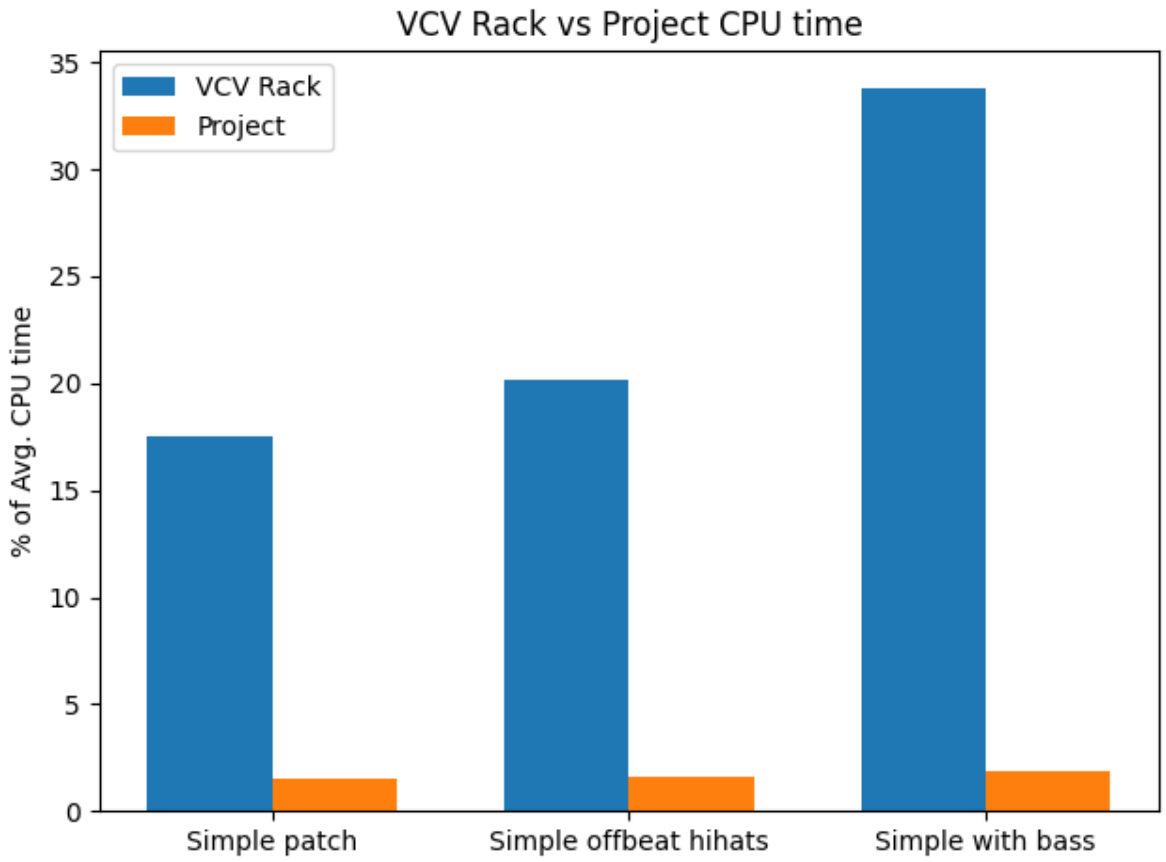

Figure 9: Charted performance assessment results

The results from this study confidently show that the project solution is far more efficient than VCV Rack in the context of simple patches. This performance difference is very likely impacted by how this project solution does not need to render a graphical user interface (GIU). VCV Rack's GUI is the only way in which the user can construct a track, so it must be comprehensive and always up to date. This is a worthy trade-off for this project,

as this project is targeted towards people who are comfortable in an IDE. However, VCV must cater to a less specialised audience, many of whom will not be familiar with IDEs. Both software solutions decrease in performance as tracks become more complex.

This means that the hypothesis is true: The project solution performs similarly or more efficiently than VCV Rack. This means that developers will not be in a position where their setup is capable of working on VCV Rack but not capable of working with this project in terms of system requirements. These results are a testament to the achievements of Pyo, which is especially impressive considering the speed differences Zehra et al. (2020) have recorded.

Project Conclusion

This project aimed to allow both beginner and experienced Python developers to create custom modular synthesis modules. An exploratory analysis of background literature was conducted, which gave insight to the current advantages and disadvantages Python faces when contextualised with alternatives such as C++. This background research also shed light on Pyo, a recent work that pushes the possibilities of digital signal processing in Python and could be utilised throughout this project to reduce workload.

Specific aspects of the project were discussed to identify an appropriate project management and software development methodology. Methodology variations were identified that could reduce risk to the project.

Appropriate assessment methods for the various areas of this project were identified and an example module that could effectively demonstrate how this software could be used was discussed. The structure of the software solution was then explained and justified. The audio visualisation software was also described.

The aims of the project were assessed and its performance was measured in each area. The project was found to have achieved its audio quality and performance aims, although more data might be needed to fully confirm that the audio quality is satisfactory. The documentation for this project was found to have been simple and effective, and the ease-of-use of the software itself has promising indications. However, more analysis is needed to be able to draw more conclusions about the ease-of-use of the software created throughout this project.

Reflective Analysis

The project management planning was well done. A fair and thorough examination of the project was given, and succinct conclusions were made. Careful thought was given to how the chosen project management methodology could be tweaked to suit the project even further. This project breakdown was very beneficial for the success of the project. The methodology was stuck to throughout the project and no alterations needed to be made. If I were to do the project again, I wouldn't change this.

During the software development phase, a confusing bug was encountered and I didn't know what was causing it. The instruments in the track fell out of synchronisation after around 60 seconds or so. After creating numerous scripts to separate the instrument audio into different channels, I was able to record and analyse the audio output. Using various measurements and calculations, I was able to ascertain that for every minute of playback, the instruments would fall out of sync by 0.07s. During a 10 second audio clip (the approximate length of the audio perception test tracks), this wouldn't be noticeable, but for any real-world use of the software, it was unacceptable. I had to redesign the data flow of the program so that there was a single source of rhythm. This completely fixed the synchronisation problems. If I had more carefully considered the structure of the program earlier in development, this may not have been an issue.

I think the analysis section went well. The performance analysis was very effective, and a strong case was made for the efficiency of the project software. The audio perception tests would be more conclusive with more participants, even if the results suggested a 50-50 split. That would've added more authority to that section of the report. However, the results that were collected were a good indication of the similarity of the audio outputs, so I think that part went well overall. The ease-of-use analysis supported the project quite well but structuring that study in a way that would allow developers to actually try out the project software would've been more beneficial. However, this would have taken a lot more time to set up as I would have had to create a tutorial environment and the number of willing participants would have been lower due to the extra work involved on their part. This would be worth doing if there was more time available to me as the extra time would also allow me to find more participants. All in all, I think the analysis is a solid representation of the success of the project.

The audio visualisation was a good example to show the capabilities of the software, but it was poorly documented. If I had more time, I would have extended this part of the project to further demonstrate capabilities and produce appropriate levels of documentation so that people could be more aware of what is possible.

Word count: 8591

References

Behnel, S., Bradshaw, R., Citro, C., Dalcin, L., Seljebotn, D. (2010) Cython: The Best of Both Worlds. *Computing in Science & Engineering*, 13(2) 31-39. Available from https://ieeexplore.ieee.org/abstract/document/5582062 [accessed 15 November 2021].

Belanger, O. (2016) Pyo, the Python DSP toolbox. In: *MM '16: Proceedings of the 24th ACM international conference on Multimedia*, Amsterdam, The Netherlands, 15-19 October. New York, USA: Association of Computing Machinery, 1214-1217. Available from https://dl.acm.org/doi/abs/10.1145/2964284.2973804?casa_token=p6aKNgJJuVgAAAAA:LE9CA4-yx_0Uejqm7uzJWs_oOAgSgPKrLD1wHjWaU688ILM9R-C_E8aiA4-BInPnR1bBjphmZMCcjQ [accessed 9 November 2021].

Chan, M. (2006). Improving algorithmic music composition with machine learning. In: *9th International Conference on Music Perception and Cognition*, Alma Mater Studiorum University of Bologna, 22-26 August. Available from https://d1wqtxts1xzle7.cloudfront.net/34392696/icmpc06-with-cover-page-v2.pdf?Expires=1645642866&Signature=IO6wWERaJpPbBpG6~8M6AE8Eqdg~7FmpLtuUDeI5Xp2SToUHAz16JqKwOvJmdnVLBLhLer5w1JcNnKGnfcBL0htlsSY6QSP~GZsqhXR22rNd~MjPni8ObHlroJqmxTVGWDMyrG28HuDqadUwVZZ6JDm56Ao7kj7lVes62LDzWsVe7cgigVF9QssUdueYDIKSKV8KEOBNBzM-e~qGeYIFHb5Ggdi1EzWNyQbLuwQFu-ygFNIbNehnR3oDWfTGy~3sAgs491yNsBZjc1vBwQT5MorP--1vGApkULyWhDAg~a6kW-nk8JFDEM6rMu8VW0Bd7zIKdTUh6ldKAT-WMlYKpg__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA [accessed 24 February 2022].

Cockburn, A. (2000) Selecting a project's methodology. *IEEE Software*, 17(4), 64-71. Available from https://doi.org/10.1109/52.854070 [accessed 4 June 2022].

Collofello, J. S. and Woodfield, S. N. (1989) Evaluating the effectiveness of reliability-assurance techniques. *Journal of Systems and Software*, 9(3), 191-195. Available from https://doi.org/10.1016/0164-1212(89)90039-3 [accessed 7 June 2022].

Graf, M. (2021) An Audio-Driven System For Real-Time Music Visualisation. Available from https://arxiv.org/pdf/2106.10134 [accessed 23 February 2022].

Joslin, R. and Müller, R. (2015) Relationships between a project management methodology and project success in different project governance contexts. *International Journal of Project Management*, 33(6), 1376-1392. Available from https://doi.org/10.1016/j.ijproman.2015.03.005 [accessed 6 June 2022].

Kekre, H.B., Bhandari, N., Nair, N., Padmanabhan, P. and Bhandari, S. (2013) A Review of Audio Fingerprinting and Comparison of Algorithms. *International Journal of Computer Applications*, 70(13), 24-30. Available from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.479.873&rep=rep1&type=pdf [accessed 7 June 2022].

Mason, R. and Simon. (2017) Introductory Programming Courses in Australasia in 2016. In: *ACE '17: Proceedings of the Nineteenth Autralasian Computing Education Conference*, Geelong, Australia, 31 January - 3 February. New York, USA: Association for Computing Machinery, 81-89. Available from https://dl.acm.org/doi/10.1145/3013499.3013512 [accessed 8 November 2021].

Orlov, E.V., Rogulenko, T.M., Smolyakov, O.A., Oshovskaya, N.V., Zvorykina, T.I., Rostanets, V.G. and Dyundik, E.P. (2021) Comparative Analysis of the Use of Kanban and Scrum Methodologies in IT Projects. *Universal Journal of Accounting and Finance*, 9(4), 693-700. Available from https://doi.org/10.13189/ujaf.2021.090415 [accessed 6 June 2022].

Reason Studios (undated) *Learn about Reason Studios: makers of Reason and more.* Available from https://www.reasonstudios.com/about [accessed 7 November 2021].

Sweigart, A. (2012) *Making games with Python & Pygame* [ebook]. Available from http://inventwithpython.com/makinggames.pdf [accessed 23 February 2022].

Tomasi, S., Schuff, D., Turetken, O. (2018) Understanding novelty: how task structure and tool familiarity moderate performance. *Behaviour & Information Technology*, 37(4), 406-418. Available from https://doi.org/10.1080/0144929X.2018.1441325 [accessed 8 June 2022].

VCV (undated) *VCV Manual - About VCV.* Available from https://vcvrack.com/manual/About [accessed 7 November 2021].

Zahra, F., Javed, M., Khan, D. and Pasha, M. (2020) Comparative Analysis of C++ and Python in Terms of Memory and Time [pre-print]. Available from https://www.preprints.org/manuscript/202012.0516/v1 [accessed 8 November 2021].

Appendix

VCV's Plugin Development Documentation
https://vcvrack.com/manual/PluginDevelopmentTutorial

Project Code
https://github.com/Tom-Clare/uol_final_year

Audio Visualisation example
https://github.com/Tom-Clare/uol_final_year/blob/main/pygame_example/sounds_pygame.py

Demonstration video
https://youtu.be/T2GpbmC3jq8