A computer interface for the creation of music

*Name:* Dadds, Jonathan

*Registration Number: 07744966  
Login:* jdadds

*Scheme:* BSc Computer Science

Project Abstract

In this project I set out to design and build a computer interface for music creation. The goal of the project was to make a piece of software which made it easy for a user to create simple pieces of music using a sampler and synthesiser.

The project involved designing individual graphical user interfaces for the synthesiser, sampler, main window and timeline components, and then linking those together and then interfacing them with audio.

The majority of the requirements of the project were met, however two minor requirements I set out to complete at the beginning were not completed. When it came to project management I massively underestimated the amount of time it would take to complete.

Acknowledgements

My project supervisor, Jerome Robinson for advising me on how to complete the planning stages, and for being available for advice on the project throughout.

My parents, for making sure that I started the programming section early.

Ed, Matt, Luke, Adelina, and many other friends for making sure that I actually took breaks from working and relaxed once in a while.

Richard for sparking this interest in music many years ago which led to me deciding to take on this particular project.

The various musicians who created the sounds I listened to on the many hours I worked on this project; I can’t work without sound and therefore I think they are deserving of a small mention.

A computer interface for the creation of music

**Table of Contents**

[Planning and Requirements 5](#_Toc257114588)

[Project Description 5](#_Toc257114589)

[Project Domain 5](#_Toc257114590)

[Aim and objective 6](#_Toc257114591)

[Platform 6](#_Toc257114592)

[Requirements 8](#_Toc257114593)

[Work Breakdown 9](#_Toc257114594)

[Milestone Identification 12](#_Toc257114595)

[Activity Sequencing 12](#_Toc257114596)

[Activity Sequencing (Diagram) 13](#_Toc257114597)

[Gantt Chart 14](#_Toc257114598)

[Research 16](#_Toc257114599)

[Design 19](#_Toc257114600)

[Architectural Design 19](#_Toc257114601)

[Performance Issues 20](#_Toc257114602)

[Form design 21](#_Toc257114603)

[Main Form 21](#_Toc257114604)

[Track layers 23](#_Toc257114605)

[Sampler Form 24](#_Toc257114606)

[Synthesiser Form 26](#_Toc257114607)

[Class design 28](#_Toc257114608)

[Synth Class 28](#_Toc257114609)

[Sample Class 30](#_Toc257114610)

[Sample Playback Class 32](#_Toc257114611)

[Synthesiser Playback Class 33](#_Toc257114612)

[Main Class 34](#_Toc257114613)

[Timeline classes 35](#_Toc257114614)

[Storage design 36](#_Toc257114615)

[Implementation 38](#_Toc257114616)

[Changes to original plan 38](#_Toc257114617)

[Development Stages 40](#_Toc257114618)

[Form creation 40](#_Toc257114619)

[Class creation 43](#_Toc257114633)

[Linking Forms with Data Structures 48](#_Toc257114634)

[Audio implementation 49](#_Toc257114635)

[Summary of implementation 52](#_Toc257114636)

[Testing 54](#_Toc257114637)

[Introduction 54](#_Toc257114638)

[Test strategy 54](#_Toc257114639)

[Test design 54](#_Toc257114640)

[Test results 55](#_Toc257114641)

[Function Testing 55](#_Toc257114642)

[Synthesiser Form tests 55](#_Toc257114643)

[Sampler Form tests 56](#_Toc257114644)

[Main form tests 56](#_Toc257114645)

[Audio tests 57](#_Toc257114646)

[Performance testing 57](#_Toc257114647)

[Alpha Testing 57](#_Toc257114648)

[Implementation Changes (post-testing) 58](#_Toc257114649)

[Conclusion 59](#_Toc257114650)

[An analysis of objectives and requirements 59](#_Toc257114651)

[Project Management 61](#_Toc257114652)

[Experience 63](#_Toc257114653)

[Possible improvements 64](#_Toc257114654)

[References 65](#_Toc257114655)

[Bibliography 65](#_Toc257114656)

[Glossary 66](#_Toc257114657)

[Appendixes 67](#_Toc257114658)

[CD Listings 67](#_Toc257114659)

[Function listing 67](#_Toc257114660)

[Main class 67](#_Toc257114661)

[Sampler class (UI) 69](#_Toc257114662)

[Synthesiser (UI) 75](#_Toc257114663)

[Synth (class) 76](#_Toc257114664)

[Sampler (Class) 76](#_Toc257114665)

[Sample Playback 76](#_Toc257114666)

[Synth Playback 76](#_Toc257114667)

[Additional Test Result Diagrams or necessary screenshots 76](#_Toc257114668)

[Software Listings 82](#_Toc257114669)

# Planning and Requirements

## Project Description

My project is to design and build a piece of music creation software which allows users to create simple songs using samplers and synthesisers. It is a developmental project, using advanced human computer interface techniques. My main focus of this project is to ensure that the interface used is one which is easy to use for the average user, so that someone new to computer music could sit down with the software and be creating music within an hour.

The software is to be designed primarily for the Windows operating system. Most of the development will take place inside the Windows 7 operating system, however testing will take place in XP and Vista to ensure compatibility. I intend on using C++ to develop the software. My decision to use C++ is based largely on the audio playback requiring multiple player instances to play multiple sounds at the same time, and I do not think that realistically it’s possible for me to produce an audio playback method which allows this, therefore I plan on making use on openly available audio packages for C++ which will allow me to concentrate on the interface more without having to worry about spending a huge amount of time researching, and then trying to get my program to output audio.

## Project Domain

One of the main problems with a system like this is way the software is presented to the user. This kind of software usually contains a lot of features and some of the large commercial pieces of software aren’t really accessible to new users because of this. For some pieces of software such as Logic Pro, one of the industry standard pieces of music software, there are many extensive training courses for users to learn how to use the software. I believe that it doesn’t have to be this way. When aiming software at a new user, they won’t need every feature you could possibly imagine, and they will want the software laid out in a way that makes sense quickly. My idea for this involves creating music sections and making them drag able to a main timeline, but I will show this in more detail later.

My main focus of this project is to look into how to make this as easy for the user as possible. It’s possible that during development I will come across new ideas too, as while I have used C++ in the past, I am not completely familiar with the interfacing. I will start out using Microsoft’s Visual Studio but intend on making it a bit more interesting than the standard form designer forms made in Visual Studio. I will look into other packages for C++ interfaces. Of the ones I have looked into so far, QT which has been developed by Nokia looks like it could be useful, as it utilises ActiveX controls and DirectX functionality, and can be implemented using Visual Studio as easily as using Visual Studio is to use on its own.

## Aim and objective

Aim To develop a piece of software that allows the user to create music using a combination of sampling using pre-recorded samples (for example, drum tracks), and digital synthesis.

Objectives 1) To look into ways of making user interfaces for a program with a fair level of complexity to make them simple for the user.

2) To design a human computer interface for this system.

3) To develop a user interface

4) To interface this with the audio packages

5) To test this completely and look for any possible improvements

6) Complete final report

## Platform

The software I intend on using to create this is Visual Studio. I intend on using this to allow me to move gently into the world of C++ graphical user interfaces, and because I feel a program of this size may become too much to deal with if I was to use my normal editor of choice; TextPad. The development will be done on a Windows 7 Professional machine. I have chosen this as it is the most recent Windows operating system and it is surprisingly very good and so far reliable. I am choosing Windows as for me it was between Windows and Linux, however I feel the majority of people use Windows and therefore it is best to develop for as many users as possible. The hardware of the development machine is the previous generation of Intel Quad Core processors, with 2gb of RAM and an ATI graphics card with 1gb of onboard memory. The audio is handled with an external soundcard, a Tascam 16 in – 4 out card, which should make any problems with audio processing on the computer easier to hear through the speakers. The computer should realistically be able to handle whatever I throw at it when it comes to this software; the audio capabilities and graphical capabilities are above par, which I feel is important in this type of program.

## Requirements

* The ability to create a new pattern in a loop based sampler.
  + This should allow 16 notes to a loop to allow a decent level of complexity
  + This should allow the samples to be panned from side to side.
  + The ability to change the sounds loaded into the loop based sampler.
* The ability to create sound with a synthesiser
  + The software needs to have a synthesiser module
  + It must be possible to enter notes into the software in a way which makes sense to the end user.
  + It must be possible to edit the sound created by the synthesiser module
  + It should be possible to have more than one instance of the synthesiser
* It must be possible to save the creation to be edited at a later date
* It must be possible to export the audio in a format usable by a standard computer music player (mp3, Ogg, WAV etc.).
* It must be possible to mix the tracks on the timeline.
* It must be possible to drag loops created in the loop based sampler onto a separate timeline to be matched with synthesiser tracks.

## Work Breakdown

Music Software

Planning

Design

Build

Evaluation

Initial Idea

Research

Interface

Algorithm Design

Interface

Audio

Linking GUI and code

Testing

Finish report

## 

A breakdown of the diagram on the previous page:

Planning section:

Initial idea – this phase has already been completed. In this phase I set about coming up with an idea and fine tuning the idea to make it a feasible project containing plenty of relevant computer science methods. This section took around a week, working in blocks rather than continuously working.

Research – this phase has been started and is running alongside the design stages. In this phase I look into other similar pieces of software that have been developed, commercial or non-commercial, as well as looking into audio packages which may help during the development process. I expect the bulk of this to take 14 days, however I realise this may overrun, and that it’s possible I will continue researching from time to time during the build stage.

Design section:

Interface – this phase is the most important one in my opinion. In this phase I will set about designing the interfaces for the software. I will take into consideration everything that I learned during the Human Computer Interfaces module from second year, as well as looking at other products to see how they have done it. I aim on having this done within 7 days. I have already made a start on this.

Algorithm design – In this phase I shall work out any specific algorithms which will need to be used in the development of the program. I think this will include the synthesiser section and the sampler section, as well as working out the best way in which to link time sections on the main window. I estimate that this will take about 5 days before starting the build stage, however I realise that this will continue during the build stage too.

Build section:

Interface – this phase is directly linked to the design phase interface, and can only be done upon completion of that phase. In this section I intend on building up a full, working interface. I expect all moving elements to move at this point. I don’t expect procedures to run, although I may add some to test elements. I expect this to take around 14-20 days due to the complicated nature of the interface.

Audio – In this phase I shall implement all the audio sections; loading samples, digital synthesis, all audio output. I expect this to take around 15 days to complete, although as it’s something I have never done before it potentially could take longer to complete.

Linking GUI and Audio code – In this phase I will take the code written in the audio phase and link it with the GUI code. I don’t expect this to take very long, most likely 3-5 days.

Evaluation section:

Testing – In this phase I will ensure that all the features that this program should have do work and have been implemented. I shall draw up a test plan, and then run through each of these tests to see whether the result is the same as what I expected. If there are any problems with the tests then I shall return to the build section, and will then need to start the testing phase again from scratch to ensure that no previously working features are broken. I expect this process to take 3-5 days, although if something isn’t working it could potentially be longer. I want to be very thorough when it comes to this phase.

Finish report – In this phase I will write up everything that I have done and how everything has gone. It is hard to estimate how long this will take until I have completed the other phases, but I would like to guess it will take about 14 days to complete.

## Milestone Identification

The most important milestones in my project are, in my opinion, the submissions set up by the department. I feel these are most important as, while with my own milestones I can change the times when things need to be done, with the departmental submissions the dates are fixed and therefore need to be taken seriously when all other times are worked out.

Most of my milestones in my project which aren’t predefined submissions are taken from the work breakdown. To work out the date to completion I put all my tasks/milestones into Microsoft Project and input the estimated duration, and the due date for parts which had deadlines.

Milestones (by date to be completed):

Write Initial proposal – 23/10/09  
Research into project – 19/11/09  
Produce project plan – 26/11/09  
Design Interface – 4/11/2009  
Design algorithms – 10/12/2009  
Build interface – 02/01/2010  
Implement audio section – 20/01/2010  
Test software package – 26/01/2010  
Produce Final Report – 29/03/2010  
Produce a presentation – 14/04/2010

## Activity Sequencing

Some of my activities do require previous milestones to have been reached before I can continue. The build section for example can only be started when the design section has been started. Within the build section the link audio and interface section can only be started after the audio and the interface have been created. The testing section can only start once the build section is complete. The final report outline can only be started when the testing is done, and the final report can only be started when the outline is done. This can be presented with the diagram bellow.

## Activity Sequencing (Diagram)

Project Plan Hand In  
+0days

Testing  
+5days

Final report+21 days

Implement Audio section  
+15days

Build Interface  
+20 days

Algorithm design  
+5days

Design interface  
+7days

Link audio and interface  
+3days

Total = min of 61 days

## Gantt Chart

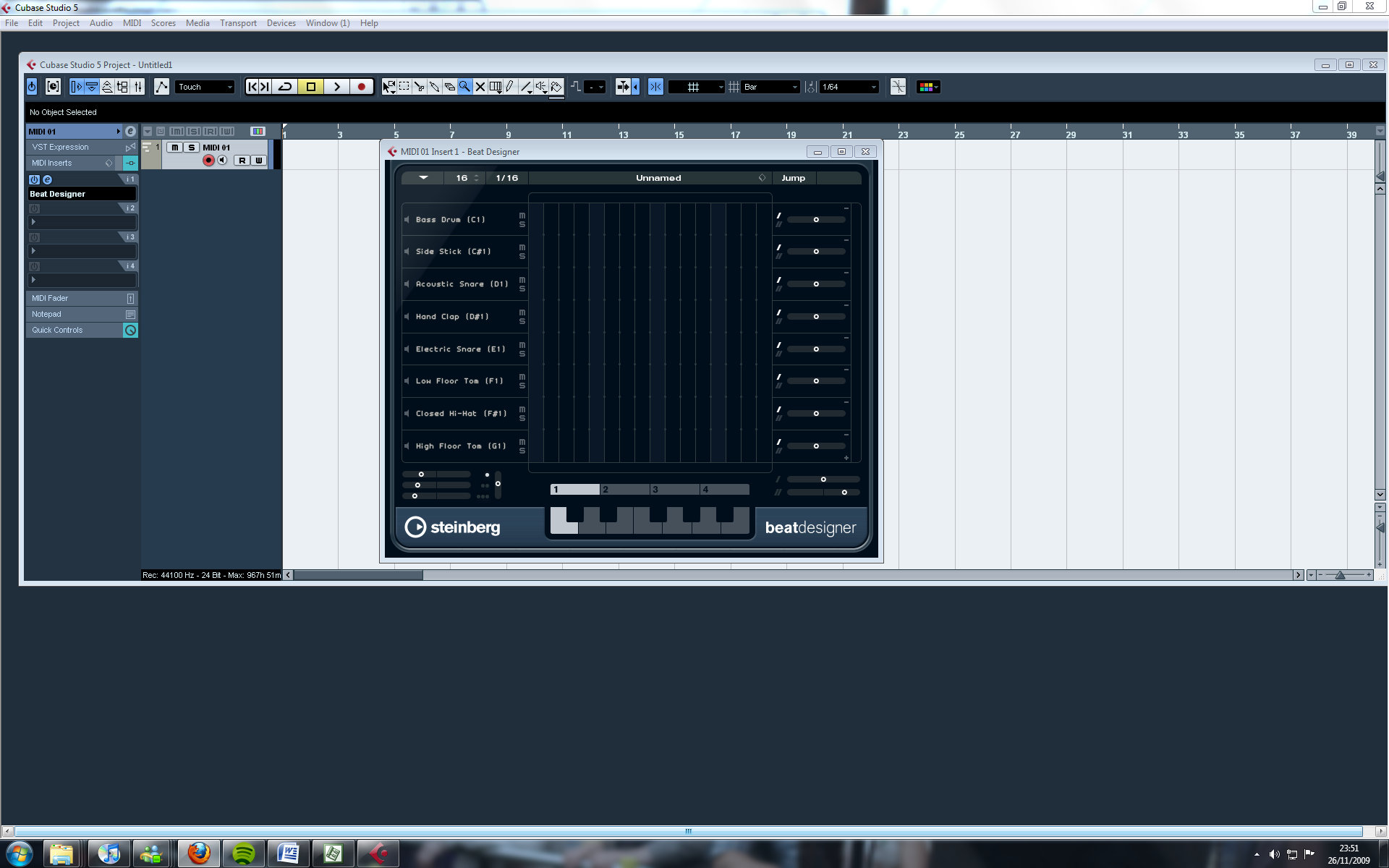
On this page is my completed Gantt chart complete with the table of data that was inputted into the MS Project.



# Research

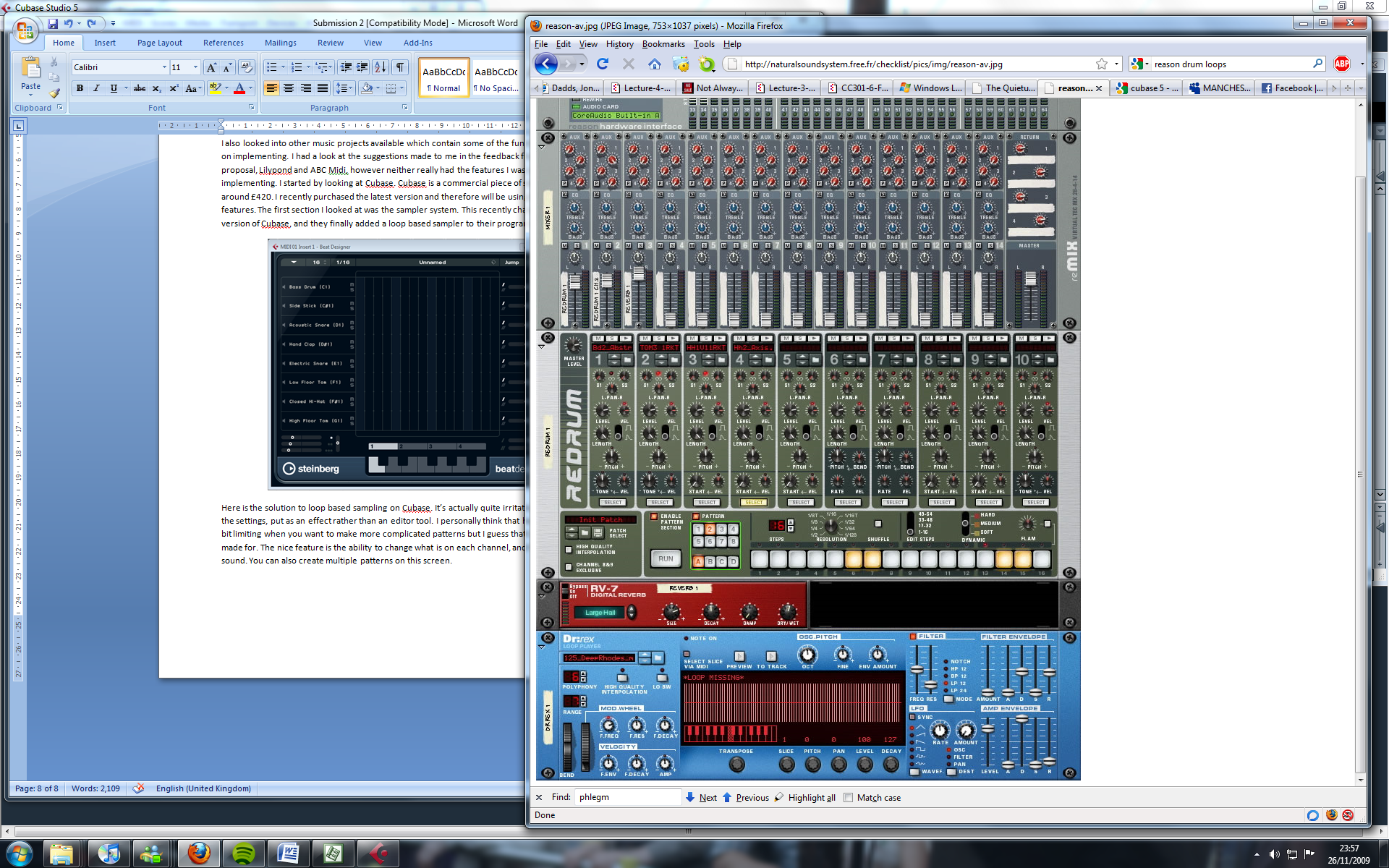
The first part that I have completed research for is which way of handling the audio would be best. I wanted to use C++ for development and I learned quite quickly that there was a package called Jack for Windows which seemingly allows for exactly what I require it to do – playing back audio samples and audio from a digital synthesiser. I started testing this however at this point I haven’t finished testing that it has all the functionality that I require from an audio package. I hope to have tested this quite soon. I heard of Jack because it has been used in a similar but more complicated program Qtractor.

I also looked into other music projects available which contain some of the functionality that I plan on implementing. I had a look at the suggestions made to me in the feedback from my initial proposal, Lilypond and ABC Midi, however neither really had the features I was planning on implementing. I started by looking at Cubase. Cubase is a commercial piece of software, which costs around £420. I recently purchased the latest version and therefore will be using that to look at the features. The first section I looked at was the sampler system. This recently changed with this version of Cubase, and they finally added a loop based sampler to their program.

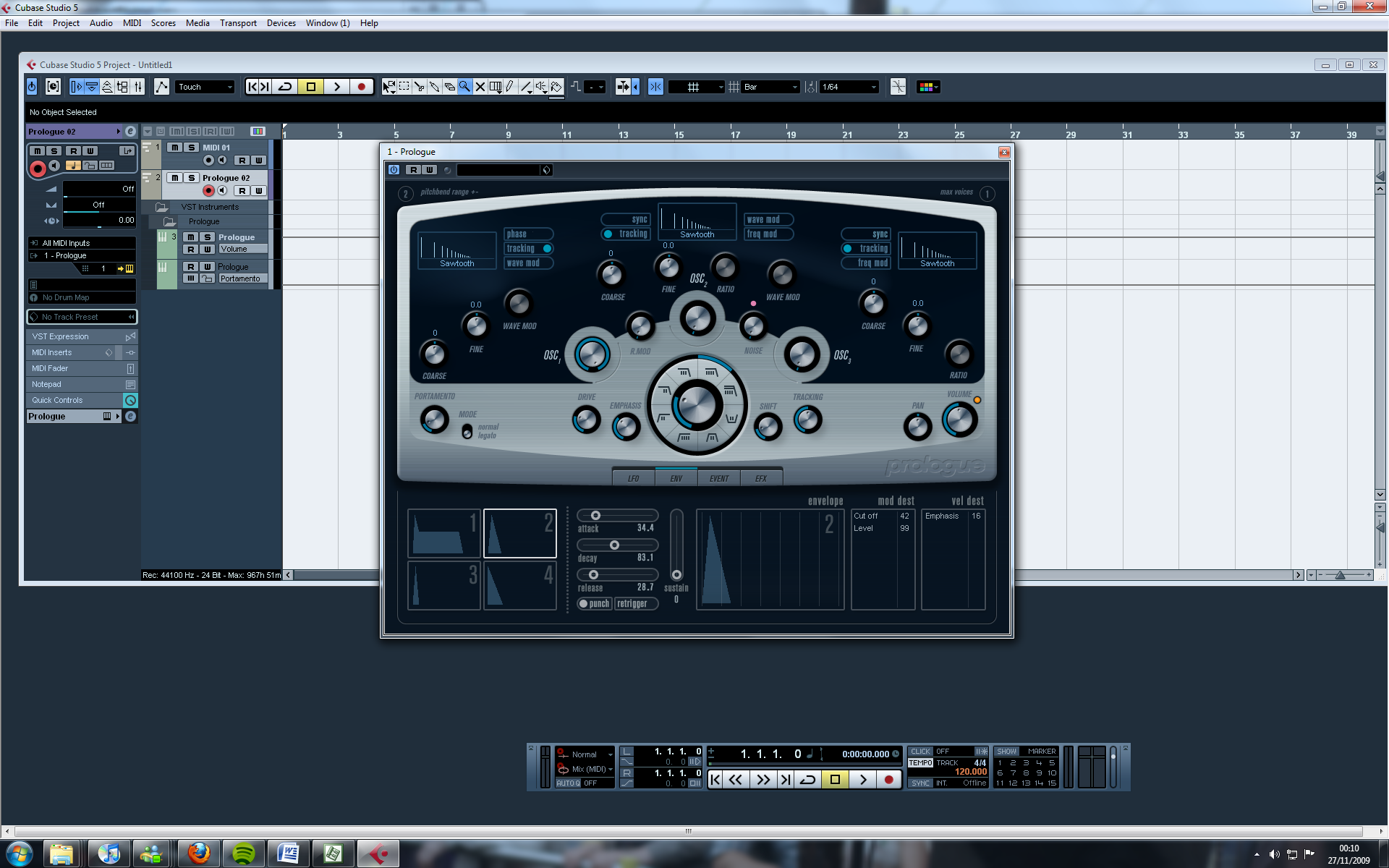


Here is the solution to loop based sampling on Cubase. It’s actually quite irritatingly tucked away in the settings, put as an effect rather than an editor tool. I personally think that having 8 channels is a bit limiting when you want to make more complicated patterns but I guess that’s not what this is made for. The nice feature is the ability to change what is on each channel, and the ability to pan the sound. You can also create multiple patterns on this screen.

The next thing I looked at was Propellorhead’s Reason. It’s slightly less than Cubase at £270. It’s method for loop based samplers takes a thing or two from the old analogue drum machines of the 70s, using individual buttons. The way this works is you choose which channel on the sampler you want to set up, and then you have a set of 16 buttons. As the program loops, the buttons get passed over one at a time. If they’re pressed in then it plays the sample.



One advantage to this is it allows quite specific patterns, and it can play 10 samples side-by-side. However, it can be fairly time consuming if you make 8 patterns with all 10 samples. I wanted to look at how this is done on Pro Tools, as a fairly respected piece of commercial software, but unfortunately have not been able to get my hands on a Mac running it, as it is now exclusively Apple Mac.

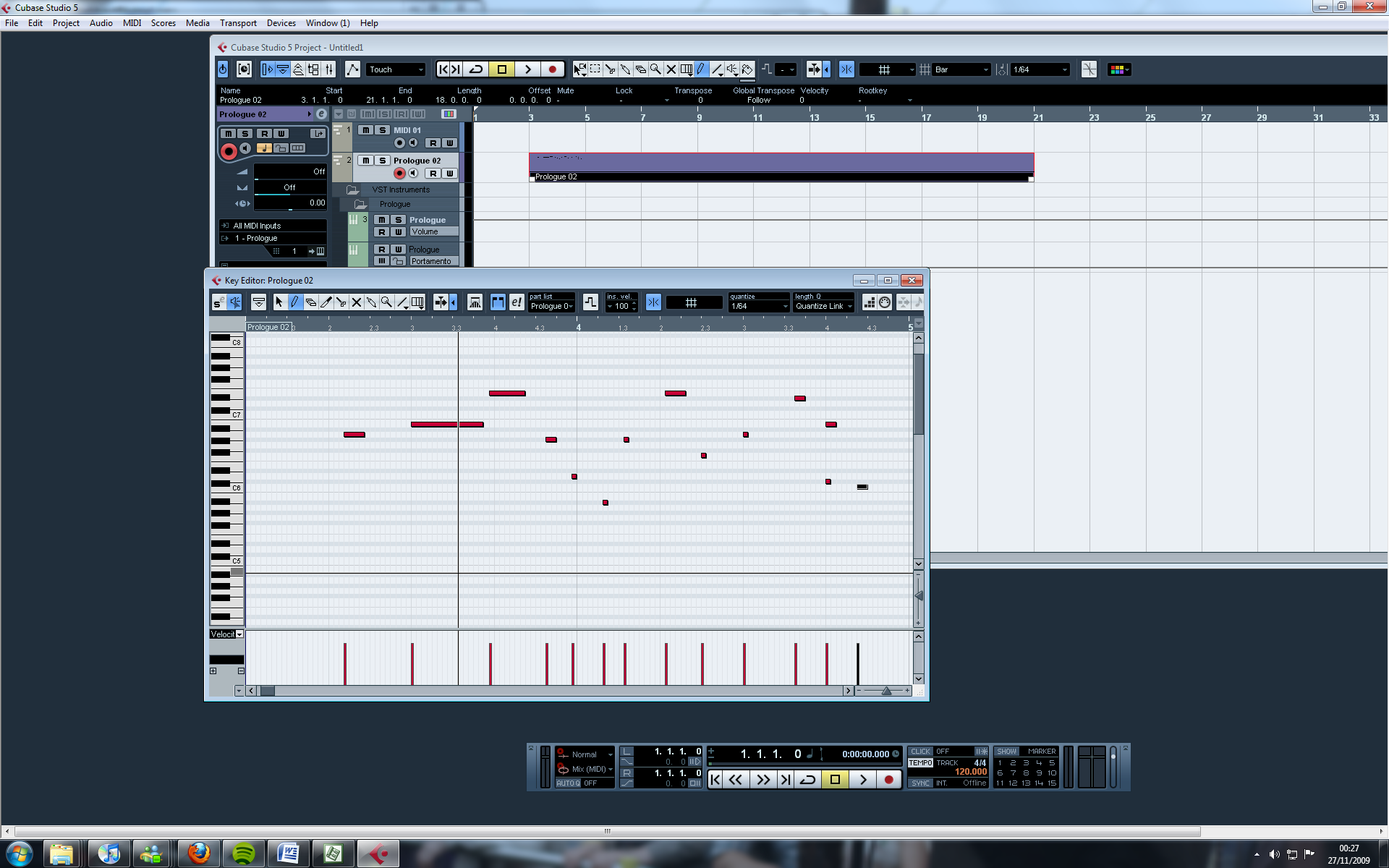
As far as synthesisers go on computers, there are two things to look at. 1) how you control the sound, 2) how you record what is happening. Here is how Cubase handles controlling what sound it makes:

As you can see, there’s an awful lot of settings and dials on this synthesiser. When I make mine I intend on making it considerably simpler, as I fear the complexity of something like this may end up confusing the target user. This one lets you manipulate the shape of the wave, the rate of just about everything on it, how deep each sound is and tons more.

As far as actually making a sound and recording it, Cubase puts the notes into an editor on different layers. These can either be entered by hitting record and then pressing notes on an external midi keyboard, or it can be done using a pen tool like shown bellow:

Keys in standard piano scale.

Notes added by user.



# Design

## Architectural Design

The first thing to consider in the design of the architecture is how the layers of the program communicate with each other. In this project my intention is for the majority of the storage to be handles by the main class, allowing for access to data by any component. When I create the synthesiser and sampler layers, I will also include a class type for each one, defining the memory which makes up one instance of the synthesiser and sampler. This is what will be stored in the main class.

The UI for both the sampler and the synthesiser will be generated from the class data.

As far as audio playback goes, there will be an instance of playback which loops, within the sampler screen, however other than that there will only be instances in the mainwindow for playback. I think the architecture can be displayed like this:

Sampler Playback (Thread)

Synthesiser UI

Sampler UI

Synth Playback (Thread)

Sampler Playback (thread)

Synthesiser Class

Sampler Class

Main Class

## Performance Issues

In a program performance is a major consideration as putting out audio on as large a scale requires a lot of the computer’s resources. One of the things to think about is the playback of samples, which will require 16 audio tracks going at once; and that’s just on one layer! The user can assign up to 30 sampler tracks to their timeline, and that means 480 audio clips could effectively be played at the same time.

When designing the algorithms to start the playback this is something I must take into consideration. While there is not an easy way of reducing the amount of resources required by the program, the amount of processing done between starting a track layer playing must be minimal, to ensure that delays from additional processing are not added.

Because the audio playback needs to run in the background, the playback classes need to be created as separate threads to ensure that when they run they don’t cause the user interface to become unstable. It also allows for better performance on multi-core computers as the more cores the machine has, the more threads that can be handled at the same time.

## Form design

### Main Form

With the main form, the first thing to think about is what features it must have. First of all, it must be able to open a new synthesiser, and edit an old one. It must also be able to open a new sampler, and edit an old one.

The main feature however, on the main form has to be the timeline and playback controls. The timeline has to accept multiple tracks containing either samplers or synthesisers. It must be possible to have a total of 30 layers on the screen. It must be possible to assign a synthesiser to each bar on a synthesiser track, and it must be possible to select a beat on each one.

Here is an initial design for the main form.

Time Signature:

Beat:

Bar:

BPM:

(Synth/Sampler area)

(timeline area)

stop

play

File insert synthesiser sampler help

Here are the details about what each component on the diagram does:

* Play Button: Starts playback of the timeline elements.
* Stop Button: Kills the playback of the timeline elements.
* Bar label: Contains the value of the currently playing bar.
* Beat label: contains the value of the currently playing beat.
* BPM label: allows the user to set the BPM of the project
* Time Signature label: allows the user to set the time signature of the project.
* Timeline area: This area will contain instances of synthesiser and/or sampler layers (these will be shown later).
* Synth/Sampler area: this will contain a list of synthesiser and sampler blocks to be dragged to the timeline area.
* Menu block (at top). This will contain the following elements:
  + File menu: this contains these commands:
    - New: Resets the main class to start a new project
    - Save: Shows a file name dialogue and allows the user to save the project.
    - Open: Shows a file name dialogue and allows the user to open the project.
    - Exit: closes the program
  + Insert menu:
    - Synthesiser: adds a new synthesiser track to the timeline.
    - Sampler: adds a new sampler track to the timeline.
  + Synthesiser menu:
    - New Synthesiser: creates a new synthesiser element and opens the synthesiser module.
    - Edit old synthesiser: Asks the user to choose an existing synth and then open the synthesiser module to edit this
    - Edit Synthesiser Settings: Allows the user to change the settings for the sound the synthesiser makes.
  + Sampler menu:
    - New Sampler: Creates a new sampler element and opens the sampler module
    - Edit old sampler: Asks the user to choose an existing sampler and then open the sampler module to edit it.
  + Help menu:
    - About: show the about screen.

### Track layers

My initial idea for a track layer was to use a drag and drop interface to move sample blocks and synthesiser blocks from the synth/sampler area, to the tracks on the timeline. With this in mind here was my original track layer idea:

Sampler

[image]

Draggable area.

However, after considering this for a while, I decided that, as I have no background in C++ UI programming, it would be best to have a second idea too, one which I thought would be possible using the UI constructors within the IDE. I came up with this design as a backup:

Sampler

[image]

Beat 4

Sample Block 0.

In this diagram a tab system is used to select which beat to edit and then a drop down list allows you to select which block to assign to that beat. I feel this would still be very good from an HCI point of view as it allows the user to clearly see which elements they have selected.

### Sampler Form

With the sampler form, the user must be able to select which beats are going to play at what time. To do this, each channel needs to have 16 options, to represent each of the 16 beats. I want it to be possible to assign either a pan or a volume change to each beat, so I will make these from dials, which allow a value to be used, rather than just a Boolean.

It must have a way off assigning a new file to a channel, so that the user can change the sample files.

The form must have a play button and a progress bar, to allow the user to test their loops before using them in the project. There will be 16 channel strips, each consisting of 16 dials, a filename box and a load button to apply the new file.

Here is my initial design for the sampler form:

Title

Play

Filename for channel

Load

Here are the details about what each component on the diagram does:

* Title block: Allows the user to set a title for the block
* Play button: Allows the user to play the loop – triggers audio playback
* Filename for channel: displays, and lets the user edit the channels file.
* Load button: Sets the sample to the file given by the user
* Dials: lets the user set the value of each beat of each channel
* Progress bar: shows the user how far through the loop they are when playing back.

### Synthesiser Form

In a similar way to the sampler form, the synthesiser form must allow the user to choose which note, volume, attack, sustain and release should be assigned to each beat in a set of 16 beats. It must be possible to choose from multiple octaves.

Unlike the Sampler unit, there is not much need for a playback system within the unit, as there are not too many occasions where looping the synth will be necessary, unlike the sampler class. Here is my initial design idea:

Volume

Attack

Sustain

Release

Low

Middle

High

Next

Previous

In this design my idea was that for each note they would press a key on the keyboard, then set the volume, the attack, the sustain and the release. They could scroll through 16 beats like this, setting it for each beat.

While I think this is an aesthetically pleasing design, I thought it may be a bit too complicated for the average user to use. It lays out the notes in an obvious way, but clicking on one key to set the value of the note is a weird way to do it, not a way that I have seen before. It makes sense using a keyboard when you can record the values being pressed by the user and put them back, as used by programs such as Cubase and Reason, however setting a value I thought it would be better to use a different method. My second design is bellow:

Previous

Next

**3**

Volume

Sustain

Attack

Release

C#2

In this design, the note is set using a dial in the middle, which shows the note in the text box above as the user spins the wheel. This has the advantage of being different to any of the methods used by other programs for when they record the users input, and therefore the user will not confuse this method of synthesiser input to the methods used by commercial software. It also has quite a low profile, and doesn’t look daunting to the user.

## Class design

As a lot of data needs to be stored, it makes a lot of sense to organise this into multiple classes, along with methods relating to that class.

As the architectural design section showed, there will need to be classes for the synthesiser, the sampler, and the audio modules. There will also need to be a main class for the main window, which will store references to various classes.

Here are my initial designs for each class.

### Synth Class

The main things which need to be stored within the synthesiser class are the attributes for the synthesiser. There are 16 bars, we need to store 16 instances of the data. The data fields which needs to be stored are note, volume, attack, sustain, release, and whether to hold the previous note. I will store this data inside an array of integer values. To access this I will write a selection of methods to set, and get, the values from the array.

class synth  
{  
public:  
 synth()  
 int getNote(int position)  
 int getVolume(int position)  
 int getAttack(int position)  
 int getSustain(int position)  
 int getRelease(int position)  
 bool getCont(int position)  
 void setNote(int position, int note)  
 void setVolume(int position, int volume)  
 void setAttack(int position, int attack)  
 void setSustain(int position, int sustain)  
 void setRelease(int position, int release)  
 void setCont(int position, bool cont)  
private:  
 int attributes[16][6]

}

The synth() method is the constructor which will set the values in the attributes table to 0 so that the attributes table is reset ready for the form to display it’s content – if it wasn’t initialised there is a chance it would display random figures.

The getAttribute functions take a position and return the value in the attributes array as an integer. The setAttribute functions take a position and a value and set the attribute array at that position to the integer value given by the user.

### Sample Class

Each of the dials on the sampler user interface will contain an integer value, which needs to be stored in the sampler class. There is 16 beats, and 16 channels to be stored so we need 256 integer values. The best way to store this is by channel, so a 3-Dimentional array is almost certainly the best way to store this. The other information which needs to be stored is a set of filenames, one for each channel.

As far as functions go, it needs to have a setAttribute method which changes array values when called, a getAttribute method which returns the value stored in a position in the array, and a setFile which changes the filename variable.

Here is my initial design for the sampler class:

class sample  
{  
public:  
 sample()  
 void setAttribute(int row, int beat, int att)  
 int getAttribute(int row, int beat)  
 void setFile(int sampleNo, const char\* location)  
private:  
 int attributes[16][16]  
 char [] sampleLocation1  
 char [] sampleLocation2  
 char [] sampleLocation3  
 char [] sampleLocation4  
 char [] sampleLocation5  
 char [] sampleLocation6  
 char [] sampleLocation7  
 char [] sampleLocation8  
 char [] sampleLocation9  
 char [] sampleLocation10  
 char [] sampleLocation11  
 char [] sampleLocation12  
 char [] sampleLocation13  
 char [] sampleLocation14  
 char [] sampleLocation15  
 char [] sampleLocation16  
}

The constructor in the class will set all the attributes to 0 in the array, and will set all the file locations to a default set of samples, so that the user has drum sounds as default.

### Sample Playback Class

As I have not implemented audio playback of any kind before, the design for this class will mostly consist of what I believe will need to be stored within the class, as opposed to specifying exact methods for doing this, as these could quite easily change.

The class needs to be a thread, as it needs to run in parallel with the UI and any other threads, it should not stop other aspects of the program from running.

As far as variable storage goes, it needs to contain an instance of the sample class, a double to store the BPM and an integer to store how far through the loop it is. It should also contain a timer element to ensure the sample plays in the correct time.

For methods, it must have a constructor, which takes an element of the sample class, and then moves that into the sample variable within the class. It must contain a run method, to start it, and a stop method, to end it. It should have a getPosition method, which allows the owner to find out where in the loop process the playback currently is. It should have a setBPM method to allow the owner to set the BPM as needed.

Here is my initial design for the sample playback class:

class samplePlayback : Thread

{  
 public:  
 samplePlayback (sample & aSample)  
 void run()  
 void stop()  
 int getPosition()  
 void setBPM(double b)

private:  
 double bpm  
 sample s  
 int position  
 Timer \*beatCount

}

### Synthesiser Playback Class

Like I mentioned in the synthesiser playback class, I have never programmed any audio playback in the past, and therefore this design will consist of what data I feel needs to be stored within the class, and what methods it requires, although the methods will almost certainly change during the development stage.

This class needs to run as a thread, as it must not interfere with the running of the UI or other threads running. When it starts to run the method which called it must continue running after this thread has started.

The playback class would need a variable of type synth class, which would be used to store the synth. It would also need to store the BPM, and the position through the loop.

The methods this class would need would be a constructor, which would take an argument of type synth class. There would then be a run method to start it, and a stop method to stop the thread. It would need a set BPM method so that the BPM can be set to the projects speed.

class synthPlayback : Thread  
{  
public:  
 synthPlayback(synth & aSynth)  
 void run()  
 void setBPM(double b)  
 void stop()

private:  
 int position  
 double bpm  
 Timer \*beatCount  
 synth s  
  
}

### Main Class

The main class will be the central storage for everything created in the other elements. It also stores the timeline and the elements on the timeline.

The first thing the main store is all the synth class instances made in the synth module. I think that 200 is a reasonable maximum for this, so have made the array 200 items big. This will have a matching pointer which points the maximum value created at any time. There will also be an openSynth integer value, pointing out the position of whichever synth is currently open in the synth editor for saving purposes.

The main class will also store an array of sampler class instance which are made in the sampler module. Again, I think that 200 is a reasonable maximum so I have made the array 200 items big. There will also be two integer array pointers, one to the maximum value, and one to the open value.

As the tracks in the timeline will be created on the fly I decided that there will be an array of up to 40 tracks of each type (synthesiser or sampler). There will also be a count to keep a track of which position this is at.

The class must store the BPM which can be sent to the playback classes. It must also have pointers to instances of the playback threads of each type. Finally it needs four methods, one to save all the data, and one to open the data, as well as play and stop methods.

Here is my initial design:

class MainWindow {  
public:  
 int openSynth  
 int openSample  
 synth synthBlocks[200]  
 int synthPosition  
 sample sampleBlocks[200]  
 int samplePosition  
 sampleTrack \* sampleTracks[40]  
 int samplerTrackCount  
 synthTrack \* synthTracks[40]  
 int synthTrackCount  
private:  
 double bpm  
 int playbackBar  
 samplePlayback \* player[30]  
 synthPlayback \* aSynth[30]  
 void play()  
 void stop()  
 void saveEverything(char \* filename)  
 void openEverything(char \* filename)  
}

### Timeline classes

The final thing to consider is that each item on the timeline will effectively be stored in classes; synthTrack and samplerTrack. While this won’t store any data of the synth and sampler class types, it will store references to these, which the main class can read when it goes to play. It will need to store the reference for each block of 16 beats, so this will be an integer value.

It will require a few methods, it will have a getSynthAtBlock method, which takes an integer value as it’s argument and returns an integer. The main purpose of this is so that the main class can read the values from the array for use in the playback method.

It will also have a setSynthAtBlock which takes two integers as arguments. This will literally set the value of the block within the class. This is useful when opening a file, for example. It will also have a setNumberOfSynthBlocks method, which will set the maximum number of available synth blocks, so it can let the user know what is available to them.

class synthTrack : {  
public:  
 void setNumberOfSynthBlocks(int number)  
 int getSynthAtBlock(int block)  
 void setSynthAtBlock(int block, int synth)  
private:  
 int blockReference[300]  
 int numberOfSynths  
}

## Storage design

When it comes to saving data in my program, I think it is best to use data files, rather than a database, so that users can easily delete files they don’t need, without any risk that they are damaging a database. It also makes the files quite light, as far as file size is concerned.

The first thing to look at is what needs saving. In the main form there are various integer values for keeping track of where in the array data is and the integers storing the maximum point of an array must be saved. Then the arrays themselves need to be saved. The arrays are of various class types so the data stored within each class must be looked at and taken into consideration.

Looking through the classes, it became apparent that most of the values stored within classes are integers, with the exception of the sampler class which contains strings to store the filenames. It is assumed that each value must be stored separately in the data file, as arrays can not be written to a data file in one line.

There are some classes which use pointers, and therefore only data from those which have already been created should be exported. As it needs to be read in the same way, the pointers must be exported to the file before the array; otherwise the program will not know how to handle the data stored in the file correctly.

Here is my first design for how the data will be saved to a file:

1. Synthesiser Position pointer
2. Sampler Position Pointer
3. The tempo
4. The data stored in all the synthesisers in the array
   1. The array starting at 0,0 right through to 16,6, putting the value out as an integer to the file.
5. The data stored in all the samplers in the array
   1. The array starting at 0,0 right through to 16,16, putting the values to the file as integers
   2. Whenever the second value reaches 16, all the file locations will be put out, and then the second value will be set as 0 and the first value will increment
6. The synthesiser track position pointer
7. The sampler track position pointer
8. The data stored in the synthesiser tracks (each track up to the pointer value)
   1. The number of beats
   2. The number of available synth blocks
   3. The values stored at each beat
9. The data stored in the sampler tracks (each track upto the pointer value)
   1. The number of beats
   2. The number of available sampler blocks
   3. The values stored at each beat

# Implementation

## Changes to original plan

The biggest change I made early on in the development process was to change the IDE and packages that I used to develop the program. In the original project plan, I specified that I was planning on using Microsoft’s Visual C++ as my development environment, to assist with the development of the graphical user interface. When looking at similar types of program I discovered a program called QTractor, a linux based audio creation package. While this piece of software is a lot more complicated than I had planned my program being, it mentioned that it had made use of the QT4 framework. I decided to have a look into this, considering that it could possibly be helpful in the creation of my project.

Upon researching the QT framework, I discovered that it was a development environment and a class library, owned by Nokia, but originally developed by Trolltech. It is designed to make it easy to move programs between operating systems, developing a program once rather than having to change the majority of code to use it on a different platform.

QT includes quite a few packages which will be invaluable in the development of my project too. The first thing I noticed is the inclusion of a dial component for the UI. As my project relies on a large number of dials components, this will allow me to avoid making this component from scratch. The other component which will prove invaluable is the QAudio class, a built in QT component which allows for easier access to the audio output. It makes alot more sense to use pre-developed audio output components as the amount of work required for me to develop a class to connect to the audio drivers and output sound would be too much for this project. There is more on the implementation of audio in the Audio Implementation section later in this document.

When it came to the synthesiser section I will admit that coding the audio output was alot more difficult that I had believed it would be, when it came to it, I made use of the RTAudio packages for sound output, and the STK Sound package for creating the sine wave. The STK was developed by Perry R. Cook and Gary P. Scavone, with RTAudio developed by Gary P. Scavone. Here is what they have to say about RTAudio:

*“****RtAudio*** *is a set of C++ classes that provide a common API (Application Programming Interface) for realtime audio input/output across Linux, Macintosh OS-X and Windows (DirectSound and ASIO) operating systems.* ***RtAudio*** *significantly simplifies the process of interacting with computer audio hardware. It was designed with the following objectives:*

* *object-oriented C++ design*
* *simple, common API across all supported platforms*
* *only one source and two header files for easy inclusion in programming projects*
* *allow simultaneous multi-api support*
* *support dynamic connection of devices*
* *provide extensive audio device parameter control*
* *allow audio device capability probing*
* *automatic internal conversion for data format, channel number compensation, (de)interleaving, and byte-swapping “* (Scavone)

RTAudio sets a method which runs whenever the thread is executed which causes the function which defines the wave (in this case the sine wave function) to execute and create the wave, which is added to the buffer, which is then sent to the soundcard.

The downside of the late change to use RTAudio/STK as my synthesiser is that I did not manage to program in the Volume/Attack/Sustain/Release methods to the synthesiser sound. The dials on the ui have been made hidden to avoid the user inputting attributes which will do nothing. This is something which if I had more time I would like to get working, but unfortunatly I did not have enough time to get this implemented at this stage. It would require more research into how I would do this, but it is likely that the original sine wave would have to be scrapped as that is a continuous oscillation, which does not degrade over time.

## Development Stages

Form creation

The first stage in the development process was to create the user interface. I started with the main form. I created this in the QT IDE designer, using the drag and drop interface. I set up a menu bar with a file menu, a view menu, a new track menu, a sampler menu, a synthesiser menu, a preferences menu and a help menu.

To begin with the main form contained a play button, a pause button, a stop button, two counters to display the beat count and the bar count, a box to enter the time signature, and a beat to enter the tempo. I also added a main timeline area, which allows you to add widgets (in this case the sampler and synthesiser tracks). I then added a drop down list, with the intention of items being dragable from the drop down list to the tracks.

The next unit I developed was the synthesiser form. I went for the second design in my original form designs, as I decided the keyboard feature would most likely be confusing for the user, as usually a keyboard input for notes would allow the user to push one note, then instantly press another, and I feel it would be frustrating for the user if they attempted to do this. I started by adding the numerical display to the top of the form, and added the first and previous buttons. I then created 4 small dials and one large dial.

The final main form I created was the sampler form. The sampler form was one which contained the most elements, and therefore took to longest to build in the IDE. There were 256 dials to place, as well as the text boxes to contain the file names, 16 load sample buttons, a play button and a progress bar. The boxes were aligned carefully with a grid. The dials were a component I thought I would struggle with, but the QT widget library had a dial element built in, which works fine for this unit. It makes sense to use the freely available item in the IDE than to create a new one from scratch, as this would have been very time consuming.

When I started trying to program the main forms user interaction with the tracking, I realised that implementing a drag and drop system would be considerably more difficult than I had originally anticipated. To implement the drag and drop system I would have had to make it so that dragging an element from the drop down list created a new block, which could then be dropped anywhere on a timeline. The timeline itself would be a very complicated element to program, as every time a new element was dragged over the timeline and released, it would need to:

Ensure there was no block currently present in the space the user was dropping the block (this includes checking at both the beginning and end beat to ensure there is no layering with other blocks.

Quantise the block – the block must fall onto a beat, no half beats.

The disadvantage to just quantising to the nearest beat is that it’s very easy to accidently end up with different tracks running at different times, making the song sound out of sync.

Because of the complications in the development and the execution of the drag and drop timeline, I decided to move to my second idea with the synthesiser and sampler tracks, and use a tab based design, where each tab can have a block assigned to it. The first advantage to this is that everything automatically quantised to the nearest bar. This means that music created will almost certainly sound in time.

This also has a major advantage when it comes to writing the code to playback synthesiser blocks and sample blocks; there will be no odd number of silent beats in the playback. This is something I did not look forward to attempting to implementing, due to an odd number of silent beats needing a completely method for playback as it would not finish the 16 beats that a standard bar would play. A silent bar would be easy to play as it would just be a normal instance of the sampler or synth class but with the attributes set to 0, however an odd one or two beats would need a way of telling the playback thread to play only the number of silent beats, and then loading the next bar.

To create the timeline and the area containing the timeline, I first tried using a scroll area element in the QT creator, but it would only let me display one track, and then when I added a new track the previous one disappeared. I then thought about how would be best to display multiple tracks. The toolbox element in the QT creator got my attention, as it allowed multiple widgets, but displayed them in a nice way which allowed easy access to any, without filling the screen.

With the way of displaying the tracks sorted, the next thing to implement the track layers themselves. The first thing I did was create illustrations to use as labels on the track. These were created pixel by pixel in Microsoft Paint with the idea of being aesthetically pleasing, but making it obvious to the user which one is which. The sampler diagram is of a reel to reel tape player, and the synthesiser image is based on an old analogue synthesiser. If the diagrams don’t make it clear, they have text labels too in the toolbox area. The main area of the track is the tab browser element, each tab of which contains a drop down list.

The way I implemented the drop down list system was to create a separate widget containing just a drop down list, and then whenever I created a new tab I put a new instance of this widget into an array, as well as displaying it within the tab. This made sure the timeline element had full access to the data stored in the drop down box, namely the value it’s pointing at, so that when playback is called it can quickly find out which block, if any, the user has selected to be played back at that point.

During the development of my program, it became apparent that I would need a save/open dialogue, and a dialogue to choose which synthesiser or sampler was to be edited when the “edit synthesiser” or “edit sampler” menu options were selected. These were incredibly simple forms to build – the save/open dialogue had a text box and an ok and cancel button, which allowed the user to enter a file name and then click ok to save or open (depending on what was clicked originally). For the edit synthesiser/sampler dialogue all it contained as a label, a drop down box consisting of a list of all available synthesiser/sampler blocks (depending on which menu called it), and an ok and cancel button.

Class creation

After creating the user interface, I moved my attention to the back-end of my program, the classes which stored the data the program would need to run.

The first class I developed was the synth class. The synth class was designed to the same specs as my original design:

class synth  
{  
public:  
 synth()  
 int getNote(int position)  
 int getVolume(int position)  
 int getAttack(int position)  
 int getSustain(int position)  
 int getRelease(int position)  
 bool getCont(int position)  
 void setNote(int position, int note)  
 void setVolume(int position, int volume)  
 void setAttack(int position, int attack)  
 void setSustain(int position, int sustain)  
 void setRelease(int position, int release)  
 void setCont(int position, bool cont)  
private:  
 int attributes[16][6]

}

I wrote the methods to retrieve and set the values within the attributes table, and tested that these work. I changed the array to “int attributes[17][7]” because I decided I would prefer to reference the values as they are (beat 1-16, rather than beat 0-15), and I decided that the amount of extra memory used and performance in processing it was minimal.

Likewise when I wrote the sampler class, I changed that variable too, just to make it simpler to follow when debugging. My change from the original design was to make the sample variables a file rather than a string, using the QT QFile type. This made it very easy to access the audio quickly, and saved creating local file variables later, which made sure the audio classes did not crash due to the file handler being deleted within memory. Here is what I ended up with after these changes:

class sample  
{  
public:  
 sample()  
 void setAttribute(int row, int beat, int att)  
 int getAttribute(int row, int beat)  
 void setFile(int sampleNo, const char\* location)  
private:  
 int attributes[17][17]  
 QFile sampleLocation1  
 QFile sampleLocation2  
 QFile sampleLocation3  
 QFile sampleLocation4  
 QFile sampleLocation5  
 QFile sampleLocation6  
 QFile sampleLocation7  
 QFile sampleLocation8  
 QFile sampleLocation9  
 QFile sampleLocation10  
 QFile sampleLocation11  
 QFile sampleLocation12  
 QFile sampleLocation13  
 QFile sampleLocation14  
 QFile sampleLocation15  
 QFile sampleLocation16  
}

The final important class to create was the main one. My original design was generally okay when it came to the setting the class up:

class MainWindow {  
public:  
 int openSynth  
 int openSample  
 synth synthBlocks[200]  
 int synthPosition  
 sample sampleBlocks[200]  
 int samplePosition  
 sampleTrack \* sampleTracks[40]  
 int samplerTrackCount  
 synthTrack \* synthTracks[40]  
 int synthTrackCount  
private:  
 double bpm  
 int playbackBar  
 samplePlayback \* player[30]  
 synthPlayback \* aSynth[30]  
 void play()  
 void stop()  
 void saveEverything(char \* filename)  
 void openEverything(char \* filename)  
}

I did, however, need to add quite a few new functions to handle button presses on the Main Window user interface. All of the main bar items needed code to be assigned to them, however this was generally quite short. The new sampler/synthesiser functions required to create a new instance of the sample or synth class in the array, and increase the pointer, then open the editor window. The playback button on the other hand, required quite a large section of code to complete:

//set all values to -1 (silent block) before reading stored data  
//sampler first….  
 for (int x=0; x<30; x++){  
 for (int y=0; y<200; y++){  
//barsOverTime is an array which holds the value in the ssampler block array which needs to be called at each block, for //each track  
 barsOverTime[x][y]=-1;  
 }  
 }  
//then read the data from the different tracks and assign it to the array  
 for (int x=0; x< samplerTrackCount; x++){  
 for (int y=0; y<sampleTracks[x]->tabPosition;y++){  
 barsOverTime[x][y]=sampleTracks[x]->getSampleAtBlock(y);  
 }  
 }  
 //then do the same with the sampler, clear the array  
 for (int x=0; x<30; x++){  
 for (int y=0; y<200; y++){  
 synthBarsOverTime[x][y]=-1;  
 }  
 }  
//then read in the data from the synthesiser tracks  
 for (int x=0; x< synthTrackCount; x++){  
 for (int y=0; y<synthTracks[x]->tabPosition;y++){  
 synthBarsOverTime[x][y]=synthTracks[x]->getSynthAtBlock(y);  
 }  
 }  
 playbackBar=0;  
//set the playback position to 0.  
 ui->lcdBar->display(playbackBar+1);  
//if it’s not a silent bar at position 0, then create a new synthesiser in the array with the synth instance chosen by the user  
//otherwise create a new instance in the array with a silent block.  
//in both cases set the bpm.  
 for (int x=0; x<synthTrackCount; x++){  
 if (synthBarsOverTime[x][playbackBar]>=0){  
 aSynth[x] = new synthAudioHandle(synthBlocks[synthBarsOverTime[x][0]]);  
 aSynth[x]->setBPM(ui->dbTempo->value());  
 } else {  
 aSynth[x] = new synthAudioHandle(silenceS);  
 aSynth[x]->setBPM(ui->dbTempo->value());  
 }  
 }  
//do the same with the sampler  
 for (int x=0; x<samplerTrackCount;x++){  
 if (barsOverTime[x][playbackBar]>=0){  
 player[x] = new samplePlaybackMain(sampleBlocks[barsOverTime[x][0]]);  
 player[x]->setBPM(ui->dbTempo->value());  
 } else {  
 player[x] = new samplePlaybackMain(silence);  
 player[x]->setBPM(ui->dbTempo->value());  
 }  
 }  
//start the sampler playback  
 for (int x=0; x<samplerTrackCount;x++){  
 player[x]->run();  
 }  
//start the synthesiser playback  
 for (int x=0; x<synthTrackCount; x++){  
 aSynth[x]->run();  
 }  
//in most instances, set it so that when the thread has finished, a method identical to this one, with an additional //“playbackBar++;” at the beginning is called.  
 if (samplerTrackCount>0){  
 connect(player[0],SIGNAL(done()),this, SLOT(samplerPlaybackComplete()));  
 connect(player[0],SIGNAL(posChanged(int)),this, SLOT(beatChanged(int)));  
 }  
//if theres no samplers though, send it to the same function as last time but without the sampler section  
 if ((samplerTrackCount==0) && (synthTrackCount>0)){  
 connect(aSynth[0],SIGNAL(done()),this, SLOT(synthPlaybackComplete()));  
 connect(aSynth[0],SIGNAL(posChanged(int)),this, SLOT(beatChanged(int)));  
 }  
//enable the stop button.  
 ui->pbStop->setEnabled(true);  
 ui->pushButton->setEnabled(false);

The reason why this method was written with multiple loops rather than just having one loop was due to the delay created by the extra processing required in one iteration when I tried it in one loop. Having the play methods in their own loops containing just the play method means that playback starts together.

The other complicated method was the saveEverything() and openEverything() methods, called from after pressing ok on the save or open dialogue box. This was mostly due to the amount of data being saved, although the implementation went remarkably smoothly, with it working after just a few attempts. Here is the code for the save element:

QFile file(filename);  
 file.open(QIODevice::WriteOnly);  
 QDataStream out(&file);  
 //first block is integer synthposition  
 out << synthPosition;  
 //then samplePosition  
 out << samplePosition;  
 //then the bpm...  
 qWarning()<< (double) ui->dbTempo->value();  
 out << ui->dbTempo->value();  
 //then 200 synths...  
 for (int x=0; x<200; x++){  
 //the synth data is stored in a 17x7 array so we can output this easily  
 for (int sx=0; sx<17; sx++){  
 for (int sy=0; sy<7; sy++){  
 out << synthBlocks[x].attributes[sx][sy];  
 }  
 }  
  
 }  
 //then 200 samplers...  
 for (int x=0; x<200; x++){  
 //the main data is a 17x17 array  
 for (int sx=0; sx<17; sx++){  
 for (int sy=0; sy<17; sy++){  
 out << sampleBlocks[x].attributes[sx][sy];  
 }  
 }  
 //but then the samplers have track names too...  
 out << (QString) sampleBlocks[x].sampleLocation1.fileName();  
 out << (QString) sampleBlocks[x].sampleLocation2.fileName();  
 out << (QString) sampleBlocks[x].sampleLocation3.fileName();  
 out << (QString) sampleBlocks[x].sampleLocation4.fileName();  
 out << (QString) sampleBlocks[x].sampleLocation5.fileName();  
 out << (QString) sampleBlocks[x].sampleLocation6.fileName();  
 out << (QString) sampleBlocks[x].sampleLocation7.fileName();  
 out << (QString) sampleBlocks[x].sampleLocation8.fileName();  
 out << (QString) sampleBlocks[x].sampleLocation9.fileName();  
 out << (QString) sampleBlocks[x].sampleLocation10.fileName();  
 out << (QString) sampleBlocks[x].sampleLocation11.fileName();  
 out << (QString) sampleBlocks[x].sampleLocation12.fileName();  
 out << (QString) sampleBlocks[x].sampleLocation13.fileName();  
 out << (QString) sampleBlocks[x].sampleLocation14.fileName();  
 out << (QString) sampleBlocks[x].sampleLocation15.fileName();  
 out << (QString) sampleBlocks[x].sampleLocation16.fileName();  
 }  
 //then samplertrackcount  
 out << samplerTrackCount;  
 //then synthtrackcount;  
 out << synthTrackCount;  
 //then synthTracks  
 for (int x=0; x<synthTrackCount; x++){  
 //complicated structure. each track stores the tabposition,  
 //the numberofsynths and an array of up to 300 synthBlocks.  
 //the synth block contains the data we need - currentindex.  
 out << synthTracks[x]->tabPosition;  
 out << synthTracks[x]->numberOfSynths;  
 for (int y=0; y<synthTracks[x]->tabPosition; y++){  
 //get the value at the block. remember this has a -1 on it. note this when opening file.  
 out << synthTracks[x]->getSynthAtBlock(y);  
 }  
 }  
 //then sampletracks  
 for (int x=0; x<samplerTrackCount; x++){  
 //complicated structure. each track stores the tabposition,  
 //the numberofsynths and an array of up to 300 sampleBlocks.  
 //the sample block contains the data we need - currentindex.  
 out << sampleTracks[x]->tabPosition;  
 out << sampleTracks[x]->numberOfSamples;  
 for (int y=0; y<sampleTracks[x]->tabPosition; y++){  
 out << sampleTracks[x]->getSampleAtBlock(y);  
 }  
 }  
 //then openTrack and openChannels;  
 out << openTrack;  
 out << openChannels;  
 file.close();  
}

The open function is the same, with everything in the same order, with just the data going in using the “in >>” instead.

Linking Forms with Data Structures

Linking the user interface to the data structures was generally quite an easy process. On the synthesiser form all I had to call was the setAttribute methods. I created a function which used the getAttribute methods and set the UI up to match what the synth class had stored.

The main class was relatively easy to set up as I had already written the functions, so it was just a case of connecting the two together. I set up an update function for when playback is running so the thread can update the beat counter at the top of the main form.

The sampler form was a bit more annoying due to the number of elements. I wrote a main function to set the value of the synth dials, however due to the way I created the form, using the designer, I had to write a line to call this method from every single dial. This was quite time consuming, and if I had learned how to code the 16x16 grid when I developed this I would have done, however c++ user interface development was new to me at the time, and this was the only way I could do it. I have since learned that it would be easier to create a 3D array of the dial class, and positioned them using nested loops. This would have allowed me to have only one function to handle this, rather than 256.

One of the things I found incredibly useful during the part of the development where I linked one thread or window to another was the Connect feature in QT, using signals and slots. The ideas is that the when you create a thread or a window from one class, you can connect a signal on that object to a function on the one which created it. This proves incredibly useful. For example, when the audio thread runs I can set a signal called “finished();” within the thread. When the function the thread is meant to do is complete, I can add the line “emit finished();” and anything connected to it will run. This could be a function on the parent thread which retrieves information from the child thread, or anything really. I use these to retrieve position information from the playback threads (to display how far through a loop the playback is) and to save the data from the synthesiser and sampler class instances within the synthesiser and sampler windows.

Connect was a feature I discovered part way through the development phase, and appears to be unique to QT. The QT user guide says:

*“Signals and slots are used for communication between objects. The signals and slots mechanism is a central feature of Qt and probably the part that differs most from the features provided by other frameworks.”* (Trolltech)

Without these feature this would have been a lot more long winded, so I am very glad I discovered this within the QT guide.

Audio implementation

The first audio class I started working on was the sampler playback class. I took my original design, and I could see straight away that I would need to add audio playback variables. Firstly, however, I had to read up on audio output methods. I knew from looking at QT’s user guide previously that it contained a way of playing back pre-recorded audio, and had a read through and found out about the QAudioOutput class. The user guide says about this class:

*“The QAudioOutput class provides an interface for sending live audio samples to the default audio output device.*

*The QAudioOutput class provides an interface for sending live audio samples to the default output device. It is intended for use by media streaming applications that can produce a stream of 8-bit unsigned or 16-bit signed audio samples. The sound output will be mixed with audio data from other applications.”* (Trolltech)

This sounded perfect for what I wanted to implement. My first change to my designed class was to add 16 variables of QAudioOutput. I noticed that QAudioOutput required a format variable, also declared in the QT library, so I added a QAudioFormat variable to my class.

The sampler output class constructor takes a sample, and then copies the contents of the sample block into the locally declared sample block. It then sets up the format variable, which is used by all the QAudioOutput instances. The format I use is a 44100 sample rate, 16 bit sample size, pcm codec with 2 channels.

When the Run method is called the first thing it does is work out the BPM as how many milliseconds between each beat. At first it tried (60/BPM)\*1000 but found that was too slow, so I changed 1000 to 500, and that sounded perfect. When this is done, it sets up the 16 audio tracks, using the format predefined in the constructor. I then create a new timer element to the pointer in the class private variables, and set this to start with the number of milliseconds worked out from the BPM, and set it so that it runs a playback method (audioPlaybackB()) at every loop.

After completing the sampler playback, I moved onto the synthesiser playback. Synthesiser playback was something that I wasn’t looking forward to implementing, as I have always seen it as something very complicated. The first thing to look at was what audio package to use to output the wave. In the sampler class I had used QAudioOutput, however this has no way of handling a wave, only premade audio files, and using samples for a synthesiser would need various lengths of recording, and notes, and would genuinely be incredibly complicated.

I had a look for packages which would allow me to do this. The first one I came across was Jack. From the homepage:

“What is JACK?

Have you ever wanted to take the audio output of one piece of software and send it to another? How about taking the output of that same program and send it to two others, then record the result in the first program? Or maybe you're a programmer who writes real-time audio and music applications and who is looking for a cross-platform API that enables not only device sharing but also inter-application audio routing, and is incredibly easy to learn and use? If so, JACK may be what you've been looking for.” (Davis)

I had a read through the guide suggested from the jack, and decided that even outputting a simple sound would be very difficult so I kept looking to see if there were any other options. I came across RTAudio, which seemed to fit the bill nicely. According to the website:

“It was designed with the following objectives:

* object-oriented C++ design
* simple, common API across all supported platforms
* only one source and two header files for easy inclusion in programming projects
* allow simultaneous multi-api support
* support dynamic connection of devices
* provide extensive audio device parameter control
* allow audio device capability probing
* automatic internal conversion for data format, channel number compensation, (de)interleaving, and byte-swapping” (Scavone)

I had a go at implementing the RTAudio class using the 2 channel saw-tooth wave generator example on the RTAudio website, located at http://www.music.mcgill.ca/~gary/rtaudio/playback.html (date accessed 7/03/2010). After installing and adding to the project the DirectX SDK, I managed to get this to compile and run, and had it outputting a saw wave. The next step was to change this to a sine wave. While I had studied maths at A-Level, I had no idea how to implement a sine wave using code. The first method I attempted was to work off the simple sine wave code example located at Harmony Central (http://www.harmony-central.com/Computer/Programming/Code/sine.html), although as this was not written with RTAudio in mind, it took quite a few changes before it was outputting sound. Unfortunately, no matter what I changed it appeared to just output a very high pitched wave. After a day of trying to sort this out, I decided to look for an implementation of a sine wave which used RTAudio, just to make sure I could get some sound output. I was amazed to find that someone had written a guide how to do this, using a library called STK. STK was developed by the same person who developed RTAudio, Gary Scavone, with the addition of Perry Cook, and contained classes designed to get sound from the RTAudio package. The tutorials section was very useful, as it contained multiple methods for implementing a sine wave using RTAudio. I went for the one labelled “callback”, which there was a function which added information to the buffer whenever there was space in the output buffer of the sound card. This guide is located at https://ccrma.stanford.edu/software/stk/crealtime.html. I played around with this for a while and managed to get it to work after quite a few hours of programming. I decided when I had it working that this would be the method I used to output the synthesiser.

There were a few changes that had to be made to make this run when the user clicks on the play button on the main form. Firstly, it needs a timer, to move beat by beat. Secondly, the constructor needed to take an instance of the synth class. The constructor also set up an array of notes to frequency, so that the class could output the correct wave. I made use of the frequency mapping table at http://peabody.sapp.org/class/st2/lab/notehz/, which although it rounded to the nearest integer value, was accurate enough for this purpose. I entered the numbers one by one into the constructor, into an array. I then wrote the play method, which started the timer, and the timer event, which called the sine.setFrequency(double) method to set the frequency. Getting to this point from the start of programming the synthesiser sound class had taken me near 4 full days of programming to achieve, even using the classes from STK and RTAudio, so I decided at this point that it would be best not to implement the “volume, attack, sustain, release” methods, as there was still other sections to be coded, and not much longer left to do so. If I choose to continue with the project, this will be the first element I look at, as it would provide the user with a wider choice of sounds, and more flexibility to the synthesiser section. As it is, I think that the low sounds, below ~200Hz, sound very nice, however the higher sounds sound very harsh through one of the speakers I tested this on.

## Summary of implementation

My major thought about my implementation is that I greatly underestimated how long I would spend programming and debugging my project. I expected to have the programming complete by the end of January but I can see now I was being very unrealistic. I estimate that I spent a minimum of three hours a day working on this, spending considerably more time during weekends, and more time in March. I think that I didn’t realise the complexity of the problem to begin with. Whenever I had a problem with the way I was implementing something, for example the timeline and the synthesiser audio, I would spend a lot of time working on a way of solving it, testing that, and making changes. I think this worked well, although as I have said, it was very time consuming.

I am convinced that my change over to the QT development environment and library was a very wise move. I did start using Visual Studio for a week before changing over to QT, and comparatively I found that QT was much better laid out, and a much nicer IDE to work with. When I take into consideration the library included with QT, I am glad that I came across it early on.

The first change to my original design involves controlling the volume. When I set out to program the sampler class, I intended on each dial controlling the volume of the sample at that point. I had originally intended on this controlling the pan of the sample, however I decided when I started implementing that controlling the volume would have much more use to the user. However, when I came to implement this, I discovered the QAudioOutput had no method for setting the volume, which rendered this useless. At the moment the sampler dials are on if they’re above 0, and off if they sit on 0.

This was also a problem when I came to my original requirement of “It must be possible to mix the tracks on the timeline.” While it is quite possible to change the volume of the synthesiser modules on different tracks, changing the volume from the QAudioOutput method would not be possible making a mixer to control the volume completely useless. I had a read into possible ways around this and came across Phonon, a class in the QT library. This class is similar to AudioOutput, however it allows for volume mixing. It’s claims about future modifications look like they could prove invaluable to this project:

“Work in Progress

Phonon and its Qt backends, though fully functional for multimedia playback, are still under development. Functionality to come is the possibility to capture media and more processors for both music and video files.

Another important consideration is to implement support for storing media to files; i.e., not playing back media directly.

We also hope in the future to be able to support direct manipulation of media streams. This will give the programmer more freedom to manipulate streams than just through processors.

Currently, the multimedia framework supports one input source. It will be possible to include several sources. This is useful in, for example, audio mixer applications where several audio sources can be sent, processed and output as a single audio stream.”

This would make it perfect for my project. I plan on having a look at Phonon when this feature is developed and seeing how it fits into my program. I think this package could be very useful in the future for applications like mine. The final change from the original plan was the synthesisers “Volume, Attack, Sustain, Release” planned feature, a mentioned in the audio implementation section.

# Testing

## Introduction

In software development, it is important to ensure that programs are properly tested. We must go through Validation testing, to ensure that the program matches the requirements from the original specification, and defect testing, to look for any problems with the program.

## Test strategy

My main form of testing will be black-box testing, looking at the problem from the outside, and not looking at the private methods and data being held by the program. The first step will be integration testing where I will test each module and the way it communicates with the other modules. I will start by testing each join one at a time, and move up to testing all modules together, using incremental interrogation testing.

There will be function testing for each window, to ensure the windows work as expected.

I will also do some performance testing, I will make sure that the audio does not struggle when it comes to playback and that it plays in time, and I will look at the memory use to ensure that it uses a reasonable amount, and that memory use does not rapidly increase over time.

Finally I will do Alpha testing, and will get one or more potential users to have a go at using the software and see how they find it. I will take any comments into consideration for future updates of the program.

## Test design

The first set of tests I will do is function testing. I will look at every function in each form and ensure that they work as planned. First up is the synthesiser form. I will test the next and previous buttons to ensure that the user can choose any beat in the range 1-16. I will ensure that the user can not go above 16, or bellow 1. I will ensure that the dial to select a note works, and that then continue previous note tick box works, and sets the note to the same note as previous.

Next I shall look at the synthesiser form. The annoying thing about having 256 dials is that I need to test they all work, so will test each one of those to ensure that when it’s turned on, a sound plays on that beat. I will also test the load file buttons, and the playback button.

I will then look at my main form. I will check that it is possible to add new tracks, edit each track, add new beats to each track, set the BPM and playback the audio.

After completing the function testing, I will move onto integration testing. The first set of integration I will test is the sample class -> main class integration. I will test this by creating an instance of the synth class, creating a synthesiser track on the main window, and choosing the synth instance as the first bar. I will then playback this, and ensure that it plays back. I will then move onto the sampler class -> main class integration. I will do this in the same way, creating an instance of the sampler class, creating a sampler track, and playing this back. I will then test the full integration by creating both a synth and a sampler instance, and creating one of each type track on the main form. I will then play this back and ensure they play back together.

Finally, after all of this is complete, I will run Alpha testing, asking a user to have a play and getting their feedback on this.

## Test results

### Function Testing

#### Synthesiser Form tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Number | Test | Expected Outcome | Actual Outcome | Pass? |
| 1 | Press Next, ensure the beat counter changes | Beat increases | Beat increases | Y |
| 2 | Ensure when beat counter is 16, next can not be pressed | Button can not be pressed | Button disabled | Y |
| 3 | Press previous, ensure that beat counter changes | Beat decreases | Beat decreases | Y |
| 4 | Ensure when beat counter is at 1, previous can not be pressed. | Button can not be pressed | Button disabled | Y |
| 5 | Change the beat at beat one | Label changes to show new note | Label Changes | Y |
| 6 | Press next and press “continue previous beat” | Note dial should lock and note change to the same as previous | Dial locks, and note changes | Y |
| 7 | Press previous again | Note should be the same as was previously set. | Note same as previously set | Y |

#### Sampler Form tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Number | Test | Expected Outcome | Actual Outcome | Pass? |
| 8 | Does load file load in a new file | File should be the new audio sample for that channel | Works IF the audio is at 44000 sample rate, 16 bit, .wav format in 2 channel. | Y |
| 9 | Does play start the playback | Audio should start | Audio starts | Y |
| 10 | Does stop stop the playback | Audio should stop | Audio stops | Y |

The dial tests are in the Appendix, under testing.

#### Main form tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Number | Test | Expected Outcome | Actual Outcome | Pass? |
| 267 | New synth button | Open synth window | Opens window on new block | Y |
| 268 | New sampler button | Open Sampler window | Opens window on new block | Y |
| 269 | Edit Synth button | Open drop down list window | Opens window, then opens synth with correct class instance | Y |
| 270 | Edit sampler button | Open drop down list window | Opens window, then opens sampler with correct class instance | Y |
| 271 | Save Button | Save file | File saves | Y |
| 272 | Open Button | Open file and set up UI | File opens, UI returns to how it was before, however synthesiser does not open position | N |
| 273 | Create synth track button | Create a new synthesiser layer on the timeline | Track created | Y |
| 274 | Create sample track button | Create a new sampler layer on the timeline | Track created | Y |
| 275 | Add beat button (sampler) | Add a new tab the sampler layer | Beat created | Y |
| 276 | Add beat button (synth) | Add a new tab to the synth layer | Beat created | Y |
| 277 | Drop down list (sampler) | Let user choose silence, or sample | Sampler selected | Y |
| 278 | Drop down list (synth) | Let user choose silence, or synth | Sampler selected | Y |
| 279 | Set tempo (180) | Tempo changes | Tempo set | Y |
| 280 | Set tempo (20) | Tempo won’t change as it’s out of bounds | Tempo set at 60 | Y |
| 281 | Set tempo (300) | Tempo won’t change as it’s out of bounds | Won’t allow it to be input as too large. | Y |
| 282 | Play button | Playback audio | Plays audio | Y |
| 283 | Stop button | Stop Audio | Stops the drums, synthesiser stalls but stops after a few seconds (up to 10 seconds) | N |

I consider the fail on test 272 to be quite a minor problem, as the tracks still open correctly and the blocks open as they should. 283 is quite annoying, because it does look like the program has crashed, however after quite a few hours working at it, I still can’t get it to stop correctly. I will keep looking at this.

#### Audio tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Number | Test | Expected Outcome | Actual Outcome | Pass? |
| 284 | Synth alone – Realtek HD Audio | Synth Plays | Plays | Y |
| 285 | Sampler alone – Realtek Hd Audio | Sampler plays | Plays | Y |
| 286 | Both together – Realtek hd audio | Both play | Plays | Y |
| 287 | Synth alone – Tascam US1641 | Synth plays | Plays | Y |
| 288 | Sampler alone – Tascam US1641 | Sampler plays | Plays | Y |
| 289 | Both together – Realtek HD Audio | Both play | Plays | Y |

Note: the lower frequencies were produced much better with the Tascam card, I presume this is a hardware issue rather than a software issue though.

### Performance testing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Number | Test | Expected Outcome | Actual Outcome | Pass? |
| 290 | Play all 16 channels on sampler – listen for delay | Should play smoothly | Plays smoothly | Y |
| 291 | Play 5 samplers, listen for delay | Should play smoothly | Tiny delay, otherwise smooth | Y |
| 292 | Play 5 synths, listen for delay | Should play smoothly | Slight stutter on start, otherwise works fine | Y |
| 293 | Have 3 synth channels, and 3 sampler channels, check playback | Should play smoothly | Slight stutter on synth start, otherwise fine | Y |
| 294 | Have 3 synth channels, and 3 sampler channels, check memory | Should be under 500mb | 30mb | Y |

### Alpha Testing

When it came to my user Alpha testing, the first test was carried out by Edward Blackwell. The main issue Ed had was with the sampler window. He said that when setting up the sampler window, the dials should change colour when they are selected. This would make it much easier to see what had been set. I agree with Ed on this aspect, and this is something I plan to implement. The time it took for the stop button to work was also commented on. The program seems to have a delay in stopping the thread, which can be as long as 10 seconds. This is one of the bugs I really need to smooth out quickly. Generally the comments were quite positive about the user interface, with the exception of the minor change to the sampler unit.

## Implementation Changes (post-testing)

After the Alpha testing, I plan on changing the sampler class so that when the sample is selected at a beat it changes colour, making it easier for the user to see what is going on. I would like to work out what is causing the problem with the synthesiser audio playback thread, so that it can stop whenever the user clicks stop.

The open buttons rebuild of the synthesiser track UI needs looking into. I have spent a few hours on this but it doesn’t seem to work despite the save and open code being identical to the sampler code, which does work.

# Conclusion

## An analysis of objectives and requirements

First of all, lets look at the original objectives:

Objectives 1) To look into ways of making user interfaces for a program with a fair level of complexity to make them simple for the user.

2) To design a human computer interface for this system.

3) To develop a user interface

4) To interface this with the audio packages

5) To test this completely and look for any possible improvements

I feel that I completed all of these steps to a good standard. I am generally happy with the user interface, and I feel they interact with the audio packages as they should.

The next thing to look at is the original requirements:

**Requirements**

* The ability to create a new pattern in a loop based sampler.
  + This should allow 16 notes to a loop to allow a decent level of complexity
  + This should allow the samples to be panned from side to side.
  + The ability to change the sounds loaded into the loop based sampler.
* The ability to create sound with a synthesiser
  + The software needs to have a synthesiser module
  + It must be possible to enter notes into the software in a way which makes sense to the end user.
  + It must be possible to edit the sound created by the synthesiser module
  + It should be possible to have more than one instance of the synthesiser
* It must be possible to save the creation to be edited at a later date
* It must be possible to export the audio in a format usable by a standard computer music player (mp3, Ogg, WAV etc.).
* It must be possible to mix the tracks on the timeline.
* It must be possible to drag loops created in the loop based sampler onto a separate timeline to be matched with synthesiser tracks.

From the original brief I changed the first section, from allowing the user to pan the samples from side to side, to allowing the user to change the volume of individual sample instances. However, this is one requirement I could not meet. I also did not make it possible to edit the sound of the synthesiser as discussed earlier. I did not manage to export the audio, and am not sure yet how I would go about doing this.

While obviously I am annoyed that I did not manage to complete these requirements, I feel that I underestimated just how difficult this project would be to complete at the beginning. There was a lot more coding than I expected, and I spent a lot of hours on each problem I came across. Because of this I do feel that despite having not reached 3 of the requirements, I am happy with the sections I did complete. I also believe that with more time I could add these requirements, and make the changes to the parts I think need improving.

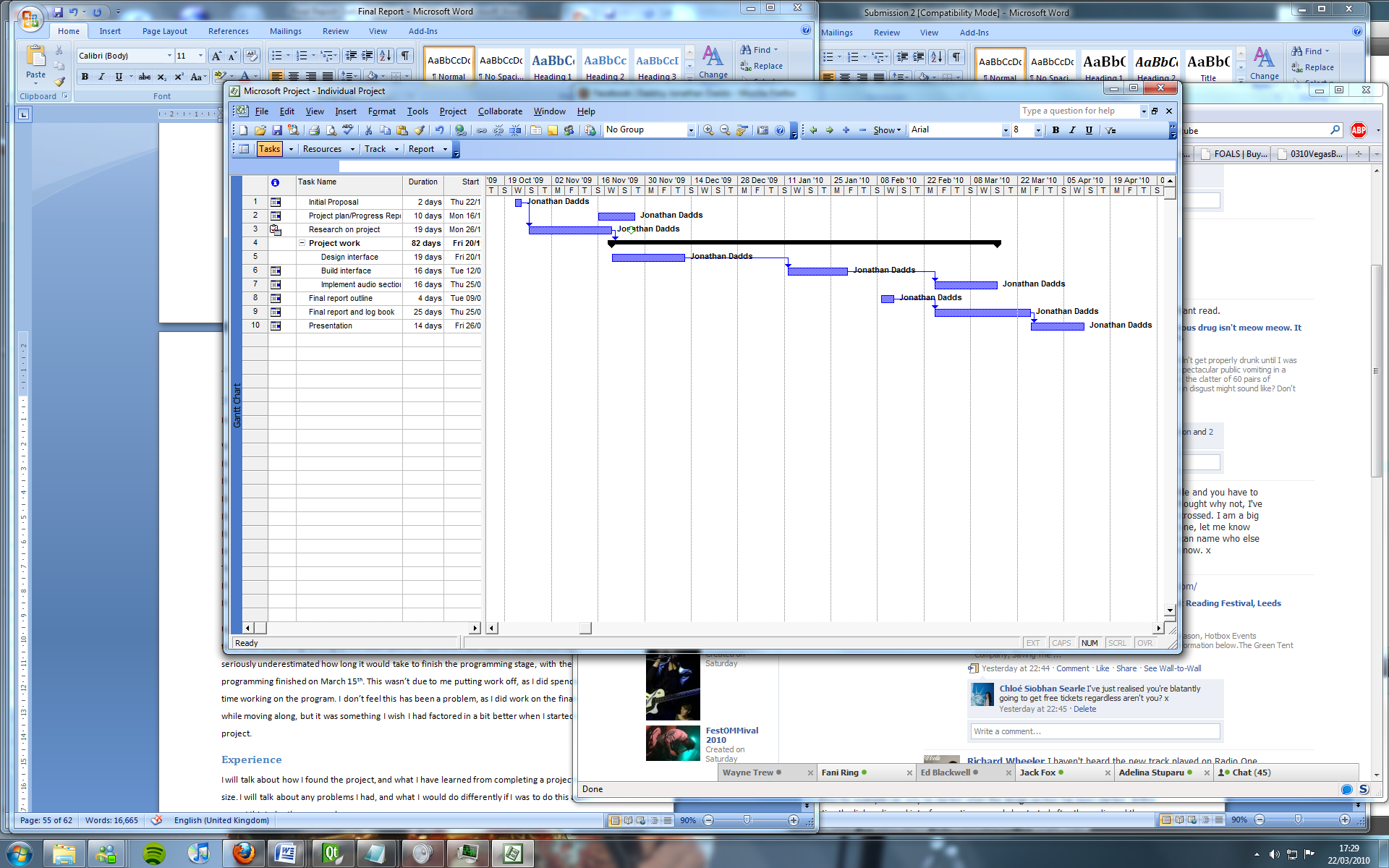
## Project Management

Here are the milestones, with dates to be completed, that I declared in my project plan:

Write Initial proposal – 23/10/09  
Research into project – 19/11/09  
Produce project plan – 26/11/09  
Design Interface – 4/11/2009  
Design algorithms – 10/12/2009  
Build interface – 02/01/2010  
Implement audio section – 20/01/2010  
Test software package – 26/01/2010  
Produce Final Report – 29/03/2010  
Produce a presentation – 14/04/2010

Up until the programming stages, build interface and implement audio section, I was doing well for time, with everything running to the dates specified. However, like I said in the previous section, I seriously underestimated how long it would take to finish the programming stage, with the programming finished on March 15th. This wasn’t due to me putting work off, as I did spend a lot of time working on the program. I don’t feel this has been a problem, as I did work on the final report while moving along, but it was something I wish I had factored in a bit better when I started with the project.

My updated Gantt chart is on the next page:



## Experience

Generally, I found the project enjoyable to complete. While it was a long process, I found it interesting and therefore managed to stay motivated a lot more than I have during other assignments. I think that my choice of topic was largely to thank for this, as I did choose something which I’ve had an interest in for many years, having used music production software for both recording, as well as for creating pieces like this program can produce, using just samples and synthesisers.

The programming was quite a good learning experience as I decided to use a language that I have not used to huge extent and try and create something quite complicated with this. In retrospect, maybe I set myself too much and would have been better off either simplifying it in the original plan or choosing a different way of doing it.

I’ve found it quite interesting to complete the project from the planning stage, right up to this, the evaluation stage. With most assignments at university we are told what to do and have a good idea how to do it so can just write code for a few hours and get it completed. It was nice to get a chance to start a project from the bottom and work our way to the top, and defiantly an experience which I hope will come in handy when I’m working.

I think that this project has massively improved my organisational skills. After three years of university, I will admit that I’ve gotten into a habit of starting assignments 2 days before they are due in, even if I don’t know how to do them, and I never really learned to do it any other way; the marks I got back were all very good, and I have had no reason to change. It’s taken this, a project of reasonable size, for me to realise that by managing my time I can achieve a lot more. For someone who’s never started a piece of coursework more than 4 days before the deadline over his university life, to start something over 3 months before it was due in was quite a big achievement. This has carried over to other modules too, with me submitting a piece of coursework 10 days before the deadline last week, something which was previously unheard of.

At parts of the assignment I did feel stressed out from the amount of work left to do. In early February I didn’t sleep properly for over a week because I was unsure how to finish large sections of the program, and I would stay up for hours thinking of solutions. I think this may be the downside to having such an interest in the project; even when you’re not working on it, you’re thinking about it, worrying about how you will complete the next task. I will admit, when the programming was finished to a standard I was happy with, I was very relieved.

## Possible improvements

The first things I would like to change would be the bugs in the program which I have struggled to fix so far. The synthesiser playback class needs to have a working stop() method, as at the minute the way the program appears that it has crashed is quite off-putting to the user.

I would then move onto improvements as mentioned in the Alpha testing section, making the dials change colour as the user selects them. I would also change the method used to output the sampler audio from QTAudioOutput to one which allows me to change the volume, making it possible for the dials on the form to control the volume, rather than sit redundant.

I would create a mixer to control the volume of each track on the timeline to make it more usable for multiple tracks. This relies on the change from QTAudioOutput to a package with a volume feature.

I would implement the “Volume, Attack, Sustain, Release” methods to the synthesiser module as was originally planned. I think after some initial research this should not be too difficult to make changes from the original classes.

In addition to fixing the missing requirements, and repairing bugs, there are a few new features I would like to implement. The first of these would be audio recording. The reason I did not make it possible to record audio in this release is due to the complexity of handling this. It would require a way of selecting which input to use as the audio source (some soundcards have more than 30 input channels so this would require a lot of coding), how many inputs to record at once, and there would need to be a way of positioning this with the sample/synthesiser blocks. I would need a way of showing the wave on the screen, and being able to cut the recording up to make better use of it. I think this would be a considerable amount of additional work.

One feature I’ve always thought would be useful from music creation software would be if there was an application that could be ran on a networked machine, or mobile device on a wireless network, which contained transport controls – something so you can press record, play, pause, stop, or rewind from a different room. This would be useful to people like me, who tend to record a lot of instruments themselves with no assistance. If, for example, you have a drum kit in a different room and you want to record it you would usually have to press record, run into the room, throw on headphones and hope you’re on time for the first beat. A mobile controller of these functions would be incredibly useful.

# References

Davis, P. (n.d.). *Jack Audio*. Retrieved 03 05, 2010

Scavone, G. P. (n.d.). *RTAudio Homepage*. Retrieved 3 5, 2010, from http://www.music.mcgill.ca/~gary/rtaudio/

Trolltech. (n.d.). *QT User Guide - Phonon Overview*. Retrieved 03 15, 2010, from http://doc.trolltech.com/4.4/phonon-overview.html

Trolltech. (n.d.). *QT User Guide - QAudioOutput*. Retrieved 02 15, 2010, from Nokia/Trolltech QT User Guide - QAudioOutput: http://qt.nokia.com/doc/qtopia4.2/qaudiooutput.html#details

# Bibliography

Carver, Richard + Tai, Kuo-Chung. *Modern Multithreading.* 2006. Wiley

Watkinson, John. *An Introduction To Digital Audio.* 1994. Focal Press

Huber, David Miles. *The Midi Manual – A Practical Guide To Midi in the Project Studio*. 2007. Focal Press.

Trolltech. *QT User Manual. Date accessed:* 05/02/2010. http://doc.trolltech.com/4.4/

Scavone, G. P. RTAudio Homepage. Date accessed: 05/03/2010. http://www.music.mcgill.ca/~gary/rtaudio/

Davis, Paul. Jack Audio Homepage. Date accessed: 05/03/2010. http://jackaudio.org/

# Glossary

Synthesiser – a way of creating a note using either digital or analogue electronics by creating a waveform.

Sampler – a device or program which plays pre-recorded samples.

Widget – a user interface component in QT (a textbox, for example, is a widget, however so is the main window containing a textbox).

BPM – Beats per minute, the number of beats the program travels through every moment which elapses.

Beat – In this context, an individual instance of sound, one drum hit, for example.

Bar – 16 beats.

Sine wave – Simple smooth wave, created using a sine function.

# Appendixes

## CD Listings

Full implementation

QT installer (for version I used to build project)

Final Report

Abstract

Readme File

## Function listing

### Main class

MainWindow(QWidget \*parent = 0);  
 ~MainWindow();  
 void changeEvent(QEvent \*e);  
 void on\_actionExit\_triggered();  
 void beatChanged(int beat);  
 void on\_actionAbout\_triggered();  
 void on\_actionNew\_triggered();  
 void on\_actionOpen\_triggered();  
 void on\_actionSave\_triggered();  
 void on\_pushButton\_3\_clicked();  
 //pause  
 void on\_pbStop\_clicked();  
 void synthPlaybackComplete();  
 void samplerPlaybackComplete();  
 void on\_pushButton\_clicked();  
 //play  
 void on\_pbEditSamplerBlock\_clicked();  
 void on\_pbEditSynthBlock\_clicked();  
 void on\_actionSampler\_triggered();  
 void on\_actionSynthesiser\_triggered();  
 void on\_actionEdit\_Synthesiser\_Block\_triggered();  
 void on\_actionEdit\_Sampler\_Block\_triggered();  
 void onSamplerSave();  
 void editSampler(int sampleNo);  
 void editSynth(int synthNo);  
 void onSynthSave();  
 void on\_actionOpen\_sampler\_triggered();  
 void on\_actionOpen\_synthesiser\_triggered();  
 void saveEverything(QString filename);  
 void openEverything(QString filename);

### Sampler class (UI)

Sampler(QWidget \*parent = 0);  
 ~Sampler();  
 void setEverything(sample\* sampler);  
 void changeEvent(QEvent \*e);   
 void saved();  
 void beenClosed();  
 void setPosition(int row, int beat, int attribute);  
 void finishedPlaying(QAudio::State state);  
 void on\_btLoad16\_clicked();  
 void on\_btLoad15\_clicked();  
 void on\_btLoad14\_clicked();  
 void on\_btLoad13\_clicked();  
 void on\_btLoad12\_clicked();  
 void on\_btLoad11\_clicked();  
 void on\_btLoad10\_clicked();  
 void on\_btLoad9\_clicked();  
 void on\_btLoad8\_clicked();  
 void on\_btLoad7\_clicked();  
 void on\_btLoad6\_clicked();  
 void on\_btLoad5\_clicked();  
 void on\_btLoad4\_clicked();  
 void on\_btLoad3\_clicked();  
 void on\_btLoad2\_clicked();  
 void on\_btLoad1\_clicked();  
 void onPosChange();  
 void on\_btPlay\_clicked();  
 void on\_dSample12Beat15\_sliderPressed();  
 void on\_dSample16Beat16\_sliderReleased();  
 void on\_dSample16Beat15\_sliderReleased();  
 void on\_dSample16Beat14\_sliderReleased();  
 void on\_dSample16Beat13\_sliderReleased();  
 void on\_dSample16Beat12\_sliderReleased();  
 void on\_dSample16Beat11\_sliderReleased();  
 void on\_dSample16Beat10\_sliderReleased();  
 void on\_dSample16Beat9\_sliderReleased();  
 void on\_dSample16Beat8\_sliderReleased();  
 void on\_dSample16Beat7\_sliderReleased();  
 void on\_dSample16Beat6\_sliderReleased();  
 void on\_dSample16Beat5\_sliderReleased();  
 void on\_dSample16Beat4\_sliderReleased();  
 void on\_dSample16Beat3\_sliderReleased();  
 void on\_dSample16Beat2\_sliderReleased();

void on\_dSample16Beat1\_sliderReleased();  
 void on\_dSample15Beat16\_sliderReleased();  
 void on\_dSample15Beat15\_sliderReleased();  
 void on\_dSample15Beat14\_sliderReleased();  
 void on\_dSample15Beat13\_sliderReleased();  
 void on\_dSample15Beat12\_sliderReleased();  
 void on\_dSample15Beat11\_sliderReleased();  
 void on\_dSample15Beat10\_sliderReleased();  
 void on\_dSample15Beat9\_sliderReleased();  
 void on\_dSample15Beat8\_sliderReleased();  
 void on\_dSample15Beat7\_sliderReleased();  
 void on\_dSample15Beat6\_sliderReleased();  
 void on\_dSample15Beat5\_sliderReleased();  
 void on\_dSample15Beat4\_sliderReleased();  
 void on\_dSample15Beat3\_sliderReleased();  
 void on\_dSample15Beat2\_sliderReleased();  
 void on\_dSample15Beat1\_sliderReleased();  
 void on\_dSample14Beat16\_sliderReleased();  
 void on\_dSample14Beat15\_sliderReleased();  
 void on\_dSample14Beat14\_sliderReleased();  
 void on\_dSample14Beat13\_sliderReleased();  
 void on\_dSample14Beat12\_sliderReleased();  
 void on\_dSample14Beat11\_sliderReleased();  
 void on\_dSample14Beat10\_sliderReleased();  
 void on\_dSample14Beat9\_sliderReleased();  
 void on\_dSample14Beat8\_sliderReleased();  
 void on\_dSample14Beat7\_sliderReleased();  
 void on\_dSample14Beat6\_sliderReleased();  
 void on\_dSample14Beat5\_sliderReleased();  
 void on\_dSample14Beat4\_sliderReleased();  
 void on\_dSample14Beat3\_sliderReleased();  
 void on\_dSample14Beat2\_sliderReleased();  
 void on\_dSample14Beat1\_sliderReleased();  
 void on\_dSample13Beat16\_sliderReleased();  
 void on\_dSample13Beat15\_sliderReleased();  
 void on\_dSample13Beat14\_sliderReleased();  
 void on\_dSample13Beat13\_sliderReleased();  
 void on\_dSample13Beat12\_sliderReleased();  
 void on\_dSample13Beat11\_sliderReleased();  
 void on\_dSample13Beat10\_sliderReleased();  
 void on\_dSample13Beat9\_sliderReleased();  
 void on\_dSample13Beat8\_sliderReleased();  
 void on\_dSample13Beat7\_sliderReleased();  
 void on\_dSample13Beat6\_sliderReleased();  
 void on\_dSample13Beat5\_sliderReleased();  
 void on\_dSample13Beat4\_sliderReleased();  
 void on\_dSample13Beat3\_sliderReleased();  
 void on\_dSample13Beat2\_sliderReleased();  
 void on\_dSample13Beat1\_sliderReleased();  
 void on\_dSample12Beat16\_sliderReleased();  
 void on\_dSample12Beat15\_sliderReleased();  
 void on\_dSample12Beat14\_sliderReleased();  
 void on\_dSample12Beat13\_sliderReleased();  
 void on\_dSample12Beat12\_sliderReleased();  
 void on\_dSample12Beat11\_sliderReleased();  
 void on\_dSample12Beat10\_sliderReleased();  
 void on\_dSample12Beat9\_sliderReleased();  
 void on\_dSample12Beat8\_sliderReleased();  
 void on\_dSample12Beat7\_sliderReleased();  
 void on\_dSample12Beat6\_sliderReleased();  
 void on\_dSample12Beat5\_sliderReleased();  
 void on\_dSample12Beat4\_sliderReleased();  
 void on\_dSample12Beat3\_sliderReleased();  
 void on\_dSample12Beat2\_sliderReleased();  
 void on\_dSample12Beat1\_sliderReleased();  
 void on\_dSample11Beat16\_sliderReleased();  
 void on\_dSample11Beat15\_sliderReleased();  
 void on\_dSample11Beat14\_sliderReleased();  
 void on\_dSample11Beat13\_sliderReleased();  
 void on\_dSample11Beat12\_sliderReleased();  
 void on\_dSample11Beat11\_sliderReleased();  
 void on\_dSample11Beat10\_sliderReleased();  
 void on\_dSample11Beat9\_sliderReleased();  
 void on\_dSample11Beat8\_sliderReleased();  
 void on\_dSample11Beat7\_sliderReleased();  
 void on\_dSample11Beat6\_sliderReleased();  
 void on\_dSample11Beat5\_sliderReleased();  
 void on\_dSample11Beat4\_sliderReleased();  
 void on\_dSample11Beat3\_sliderReleased();  
 void on\_dSample11Beat2\_sliderReleased();  
 void on\_dSample11Beat1\_sliderReleased();  
 void on\_dSample10Beat16\_sliderReleased();  
 void on\_dSample10Beat15\_sliderReleased();  
 void on\_dSample10Beat14\_sliderReleased();  
 void on\_dSample10Beat13\_sliderReleased();  
 void on\_dSample10Beat12\_sliderReleased();  
 void on\_dSample10Beat11\_sliderReleased();  
 void on\_dSample10Beat10\_sliderReleased();  
 void on\_dSample10Beat9\_sliderReleased();  
 void on\_dSample10Beat8\_sliderReleased();  
 void on\_dSample10Beat7\_sliderReleased();  
 void on\_dSample10Beat6\_sliderReleased();  
 void on\_dSample10Beat5\_sliderReleased();  
 void on\_dSample10Beat4\_sliderReleased();  
 void on\_dSample10Beat3\_sliderReleased();  
 void on\_dSample10Beat2\_sliderReleased();  
 void on\_dSample10Beat1\_sliderReleased();  
 void on\_dSample9Beat16\_sliderReleased();  
 void on\_dSample9Beat15\_sliderReleased();  
 void on\_dSample9Beat14\_sliderReleased();  
 void on\_dSample9Beat13\_sliderReleased();  
 void on\_dSample9Beat12\_sliderReleased();  
 void on\_dSample9Beat11\_sliderReleased();  
 void on\_dSample9Beat10\_sliderReleased();  
 void on\_dSample9Beat9\_sliderReleased();  
 void on\_dSample9Beat8\_sliderReleased();  
 void on\_dSample9Beat7\_sliderReleased();  
 void on\_dSample9Beat6\_sliderReleased();  
 void on\_dSample9Beat5\_sliderReleased();  
 void on\_dSample9Beat4\_sliderReleased();  
 void on\_dSample9Beat3\_sliderReleased();  
 void on\_dSample9Beat2\_sliderReleased();  
 void on\_dSample9Beat1\_sliderReleased();  
 void on\_dSample8Beat16\_sliderReleased();  
 void on\_dSample8Beat15\_sliderReleased();  
 void on\_dSample8Beat14\_sliderReleased();  
 void on\_dSample8Beat13\_sliderReleased();  
 void on\_dSample8Beat12\_sliderReleased();  
 void on\_dSample8Beat11\_sliderReleased();  
 void on\_dSample8Beat10\_sliderReleased();  
 void on\_dSample8Beat9\_sliderReleased();  
 void on\_dSample8Beat8\_sliderReleased();  
 void on\_dSample8Beat7\_sliderReleased();  
 void on\_dSample8Beat6\_sliderReleased();  
 void on\_dSample8Beat5\_sliderReleased();  
 void on\_dSample8Beat4\_sliderReleased();  
 void on\_dSample8Beat3\_sliderReleased();  
 void on\_dSample8Beat2\_sliderReleased();  
 void on\_dSample8Beat1\_sliderReleased();   
void on\_dSample7Beat16\_sliderReleased();  
 void on\_dSample7Beat15\_sliderReleased();  
 void on\_dSample7Beat14\_sliderReleased();  
 void on\_dSample7Beat13\_sliderReleased();  
 void on\_dSample7Beat12\_sliderReleased();  
 void on\_dSample7Beat11\_sliderReleased();  
 void on\_dSample7Beat10\_sliderReleased();  
 void on\_dSample7Beat9\_sliderReleased();  
 void on\_dSample7Beat8\_sliderReleased();  
 void on\_dSample7Beat7\_sliderReleased();  
 void on\_dSample7Beat6\_sliderReleased();  
 void on\_dSample7Beat5\_sliderReleased();  
 void on\_dSample7Beat4\_sliderReleased();  
 void on\_dSample7Beat3\_sliderReleased();  
 void on\_dSample7Beat2\_sliderReleased();  
 void on\_dSample7Beat1\_sliderReleased();  
 void on\_dSample6Beat16\_sliderReleased();  
 void on\_dSample6Beat15\_sliderReleased();  
 void on\_dSample6Beat14\_sliderReleased();  
 void on\_dSample6Beat13\_sliderReleased();  
 void on\_dSample6Beat12\_sliderReleased();  
 void on\_dSample6Beat11\_sliderReleased();  
 void on\_dSample6Beat10\_sliderReleased();  
 void on\_dSample6Beat9\_sliderReleased();  
 void on\_dSample6Beat8\_sliderReleased();  
 void on\_dSample6Beat7\_sliderReleased();  
 void on\_dSample6Beat6\_sliderReleased();  
 void on\_dSample6Beat5\_sliderReleased();  
 void on\_dSample6Beat4\_sliderReleased();  
 void on\_dSample6Beat2\_sliderReleased();  
 void on\_dSample6Beat1\_sliderReleased();  
 void on\_dSample5Beat15\_sliderReleased();  
 void on\_dSample5Beat14\_sliderReleased();  
 void on\_dSample5Beat13\_sliderReleased();  
 void on\_dSample5Beat12\_sliderReleased();  
 void on\_dSample5Beat11\_sliderReleased();  
 void on\_dSample5Beat10\_sliderReleased();  
 void on\_dSample5Beat9\_sliderReleased();  
 void on\_dSample5Beat8\_sliderReleased();  
 void on\_dSample5Beat7\_sliderReleased();  
 void on\_dSample5Beat6\_sliderReleased();  
 void on\_dSample5Beat5\_sliderReleased();  
 void on\_dSample5Beat4\_sliderReleased();  
 void on\_dSample5Beat3\_sliderReleased();  
 void on\_dSample5Beat2\_sliderReleased();  
 void on\_dSample5Beat1\_sliderReleased();  
 void on\_dSample4Beat16\_sliderReleased();  
 void on\_dSample4Beat15\_sliderReleased();  
 void on\_dSample4Beat14\_sliderReleased();  
 void on\_dSample4Beat13\_sliderReleased();  
 void on\_dSample4Beat12\_sliderReleased();  
 void on\_dSample4Beat11\_sliderReleased();  
 void on\_dSample4Beat10\_sliderReleased();  
 void on\_dSample4Beat9\_sliderReleased();  
 void on\_dSample4Beat8\_sliderReleased();  
 void on\_dSample4Beat7\_sliderReleased();  
 void on\_dSample4Beat6\_sliderReleased();  
 void on\_dSample4Beat5\_sliderReleased();  
 void on\_dSample4Beat4\_sliderReleased();  
 void on\_dSample4Beat3\_sliderReleased();  
 void on\_dSample4Beat2\_sliderReleased();  
 void on\_dSample4Beat1\_sliderReleased();  
 void on\_dSample3Beat16\_sliderReleased();  
 void on\_dSample3Beat15\_sliderReleased();  
 void on\_dSample3Beat14\_sliderReleased();  
 void on\_dSample3Beat13\_sliderReleased();  
 void on\_dSample3Beat12\_sliderReleased();  
 void on\_dSample3Beat11\_sliderReleased();  
 void on\_dSample3Beat10\_sliderReleased();  
 void on\_dSample3Beat9\_sliderReleased();  
 void on\_dSample3Beat8\_sliderReleased();  
 void on\_dSample3Beat7\_sliderReleased();  
 void on\_dSample3Beat6\_sliderReleased();  
 void on\_dSample3Beat5\_sliderReleased();   
 void on\_dSample3Beat4\_sliderReleased();  
 void on\_dSample3Beat3\_sliderReleased();  
 void on\_dSample3Beat2\_sliderReleased();  
 void on\_dSample3Beat1\_sliderReleased();  
 void on\_dSample2Beat16\_sliderReleased();  
 void on\_dSample2Beat15\_sliderReleased();  
 void on\_dSample2Beat14\_sliderReleased();  
 void on\_dSample2Beat13\_sliderReleased();  
 void on\_dSample2Beat12\_sliderReleased();  
 void on\_dSample2Beat11\_sliderReleased();  
 void on\_dSample2Beat10\_sliderReleased();  
 void on\_dSample2Beat9\_sliderReleased();  
 void on\_dSample2Beat8\_sliderReleased();  
 void on\_dSample2Beat7\_sliderReleased();  
 void on\_dSample2Beat6\_sliderReleased();  
 void on\_dSample2Beat5\_sliderReleased();  
 void on\_dSample2Beat4\_sliderReleased();  
 void on\_dSample2Beat3\_sliderReleased();  
 void on\_dSample2Beat2\_sliderReleased();  
 void on\_dSample2Beat1\_sliderReleased();  
 void on\_dSample1Beat16\_sliderReleased();  
 void on\_dSample1Beat15\_sliderReleased();  
 void on\_dSample1Beat14\_sliderReleased();  
 void on\_dSample1Beat13\_sliderReleased();  
 void on\_dSample1Beat12\_sliderReleased();  
 void on\_dSample1Beat11\_sliderReleased();  
 void on\_dSample1Beat10\_sliderReleased();  
 void on\_dSample1Beat9\_sliderReleased();  
 void on\_dSample1Beat8\_sliderReleased();  
 void on\_dSample1Beat7\_sliderReleased();  
 void on\_dSample1Beat6\_sliderReleased();  
 void on\_dSample1Beat5\_sliderReleased();  
 void on\_dSample1Beat4\_sliderReleased();  
 void on\_dSample1Beat3\_sliderReleased();  
 void on\_dSample1Beat2\_sliderReleased();  
 void on\_dSample1Beat1\_sliderReleased();

### Synthesiser (UI)

Synthesiser(QWidget \*parent = 0);  
 ~Synthesiser();  
 void setSlots();  
 void resetPos();  
 void changeEvent(QEvent \*e);  
 void saved();  
 void weClosed();  
 void on\_cbHold\_toggled(bool checked);  
 void on\_dRelease\_sliderMoved(int position);  
 void on\_dSustain\_sliderMoved(int position);  
 void on\_dAttack\_sliderMoved(int position);  
 void on\_dVolume\_sliderMoved(int position);  
 void on\_dVolume\_valueChanged(int value);  
 void on\_dNote\_dialReleased();  
 void on\_blPrevious\_clicked();  
 void on\_btNext\_clicked();  
 void on\_dNote\_sliderMoved(int position);

### Synth (class)

int getNote(int position);  
 int getVolume(int position);  
 int getAttack(int position);  
 int getSustain(int position);  
 int getRelease(int position);  
 bool getCont(int position);  
 void setNote(int position, int note);  
 void setVolume(int position, int volume);  
 void setAttack(int position, int attack);  
 void setSustain(int position, int sustain);  
 void setRelease(int position, int release);  
 void setCont(int position, bool cont);

### Sampler (Class)

sample();  
 void setAttribute(int row, int beat, int att);  
 int getAttribute(int row, int beat);  
 int attributes[17][17];  
 void setFile(int sampleNo, const char\* location);

### Sample Playback

samplePlaybackMain(sample & aSample);  
 virtual void run();  
 void stop();  
 void setBPM(double b);   
 void posChanged(int);  
 void done();  
 void audioPlaybackB();  
 void finishedPlaying(QAudio::State);

### Synth Playback

synthAudioHandle(synth & aSynth);  
 void makeASound();  
 virtual void run();  
 void setPitch(int pitch);  
 void setUp(synth & aSynth);  
 void setBPM(double b);  
 void stop();  
 void done();  
 void posChanged(int);  
 void pitchPerBeat();

## Additional Test Result Diagrams or necessary screenshots

Sampler Section testing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Number | Test | Expected Outcome | Actual Outcome | Pass? |
| 11 | Set dial sample 1 beat 1 | Beat should play | Plays | Y |
| 12 | Set dial sample 1 beat 2 | Beat should play | Plays | Y |
| 13 | Set dial sample 1 beat 3 | Beat should play | Plays | Y |
| 14 | Set dial sample 1 beat 4 | Beat should play | Plays | Y |
| 15 | Set dial sample 1 beat 5 | Beat should play | Plays | Y |
| 16 | Set dial sample 1 beat 6 | Beat should play | Plays | Y |
| 17 | Set dial sample 1 beat 7 | Beat should play | Plays | Y |
| 18 | Set dial sample 1 beat 8 | Beat should play | Plays | Y |
| 19 | Set dial sample 1 beat 9 | Beat should play | Plays | Y |
| 20 | Set dial sample 1 beat 10 | Beat should play | Plays | Y |
| 21 | Set dial sample 1 beat 11 | Beat should play | Plays | Y |
| 22 | Set dial sample 1 beat 12 | Beat should play | Plays | Y |
| 23 | Set dial sample 1 beat 13 | Beat should play | Plays | Y |
| 24 | Set dial sample 1 beat 14 | Beat should play | Plays | Y |
| 25 | Set dial sample 1 beat 15 | Beat should play | Plays | Y |
| 26 | Set dial sample 1 beat 16 | Beat should play | Plays | Y |
| 27 | Set dial sample 2 beat 1 | Beat should play | Plays | Y |
| 28 | Set dial sample 2 beat 2 | Beat should play | Plays | Y |
| 29 | Set dial sample 2 beat 3 | Beat should play | Plays | Y |
| 30 | Set dial sample 2 beat 4 | Beat should play | Plays | Y |
| 31 | Set dial sample 2 beat 5 | Beat should play | Plays | Y |
| 32 | Set dial sample 2 beat 6 | Beat should play | Plays | Y |
| 33 | Set dial sample 2 beat 7 | Beat should play | Plays | Y |
| 34 | Set dial sample 2 beat 8 | Beat should play | Plays | Y |
| 35 | Set dial sample 2 beat 9 | Beat should play | Plays | Y |
| 36 | Set dial sample 2 beat 10 | Beat should play | Plays | Y |
| 37 | Set dial sample 2 beat 11 | Beat should play | Plays | Y |
| 38 | Set dial sample 2 beat 12 | Beat should play | Plays | Y |
| 39 | Set dial sample 2 beat 13 | Beat should play | Plays | Y |
| 40 | Set dial sample 2 beat 14 | Beat should play | Plays | Y |
| 41 | Set dial sample 2 beat 15 | Beat should play | Plays | Y |
| 42 | Set dial sample 2 beat 16 | Beat should play | Plays | Y |
| 43 | Set dial sample 3 beat 1 | Beat should play | Plays | Y |
| 44 | Set dial sample 3 beat 2 | Beat should play | Plays | Y |
| 45 | Set dial sample 3 beat 3 | Beat should play | Plays | Y |
| 46 | Set dial sample 3 beat 4 | Beat should play | Plays | Y |
| 47 | Set dial sample 3 beat 5 | Beat should play | Plays | Y |
| 48 | Set dial sample 3 beat 6 | Beat should play | Plays | Y |
| 49 | Set dial sample 3 beat 7 | Beat should play | Plays | Y |
| 50 | Set dial sample 3 beat 8 | Beat should play | Plays | Y |
| 51 | Set dial sample 3 beat 9 | Beat should play | Plays | Y |
| 52 | Set dial sample 3 beat 10 | Beat should play | Plays | Y |
| 53 | Set dial sample 3 beat 11 | Beat should play | Plays | Y |
| 54 | Set dial sample 3 beat 12 | Beat should play | Plays | Y |
| 55 | Set dial sample 3 beat 13 | Beat should play | Plays | Y |
| 56 | Set dial sample 3 beat 14 | Beat should play | Plays | Y |
| 57 | Set dial sample 3 beat 15 | Beat should play | Plays | Y |
| 58 | Set dial sample 3 beat 16 | Beat should play | Plays | Y |
| 59 | Set dial sample 4 beat 1 | Beat should play | Plays | Y |
| 60 | Set dial sample 4 beat 2 | Beat should play | Plays | Y |
| 61 | Set dial sample 4 beat 3 | Beat should play | Plays | Y |
| 62 | Set dial sample 4 beat 4 | Beat should play | Plays | Y |
| 63 | Set dial sample 4 beat 5 | Beat should play | Plays | Y |
| 64 | Set dial sample 4 beat 6 | Beat should play | Plays | Y |
| 65 | Set dial sample 4 beat 7 | Beat should play | Plays | Y |
| 66 | Set dial sample 4 beat 8 | Beat should play | Plays | Y |
| 67 | Set dial sample 4 beat 9 | Beat should play | Plays | Y |
| 68 | Set dial sample 4 beat 10 | Beat should play | Plays | Y |
| 69 | Set dial sample 4 beat 11 | Beat should play | Plays | Y |
| 70 | Set dial sample 4 beat 12 | Beat should play | Plays | Y |
| 71 | Set dial sample 4 beat 13 | Beat should play | Plays | Y |
| 72 | Set dial sample 4 beat 14 | Beat should play | Plays | Y |
| 73 | Set dial sample 4 beat 15 | Beat should play | Plays | Y |
| 74 | Set dial sample 4 beat 16 | Beat should play | Plays | Y |
| 75 | Set dial sample 5 beat 1 | Beat should play | Plays | Y |
| 76 | Set dial sample 5 beat 2 | Beat should play | Plays | Y |
| 77 | Set dial sample 5 beat 3 | Beat should play | Plays | Y |
| 78 | Set dial sample 5 beat 4 | Beat should play | Plays | Y |
| 79 | Set dial sample 5 beat 5 | Beat should play | Plays | Y |
| 80 | Set dial sample 5 beat 6 | Beat should play | Plays | Y |
| 81 | Set dial sample 5 beat 7 | Beat should play | Plays | Y |
| 82 | Set dial sample 5 beat 8 | Beat should play | Plays | Y |
| 83 | Set dial sample 5 beat 9 | Beat should play | Plays | Y |
| 84 | Set dial sample 5 beat 10 | Beat should play | Plays | Y |
| 85 | Set dial sample 5 beat 11 | Beat should play | Plays | Y |
| 86 | Set dial sample 5 beat 12 | Beat should play | Plays | Y |
| 87 | Set dial sample 5 beat 13 | Beat should play | Plays | Y |
| 88 | Set dial sample 5 beat 14 | Beat should play | Plays | Y |
| 89 | Set dial sample 5 beat 15 | Beat should play | Plays | Y |
| 90 | Set dial sample 5 beat 16 | Beat should play | Plays | Y |
| 91 | Set dial sample 6 beat 1 | Beat should play | Plays | Y |
| 92 | Set dial sample 6 beat 2 | Beat should play | Plays | Y |
| 93 | Set dial sample 6 beat 3 | Beat should play | Plays | Y |
| 94 | Set dial sample 6 beat 4 | Beat should play | Plays | Y |
| 95 | Set dial sample 6 beat 5 | Beat should play | Plays | Y |
| 96 | Set dial sample 6 beat 6 | Beat should play | Plays | Y |
| 97 | Set dial sample 6 beat 7 | Beat should play | Plays | Y |
| 98 | Set dial sample 6 beat 8 | Beat should play | Plays | Y |
| 99 | Set dial sample 6 beat 9 | Beat should play | Plays | Y |
| 100 | Set dial sample 6 beat 10 | Beat should play | Plays | Y |
| 101 | Set dial sample 6 beat 11 | Beat should play | Plays | Y |
| 102 | Set dial sample 6 beat 12 | Beat should play | Plays | Y |
| 103 | Set dial sample 6 beat 13 | Beat should play | Plays | Y |
| 104 | Set dial sample 6 beat 14 | Beat should play | Plays | Y |
| 105 | Set dial sample 6 beat 15 | Beat should play | Plays | Y |
| 106 | Set dial sample 6 beat 16 | Beat should play | Plays | Y |
| 107 | Set dial sample 7 beat 1 | Beat should play | Plays | Y |
| 108 | Set dial sample 7 beat 2 | Beat should play | Plays | Y |
| 109 | Set dial sample 7 beat 3 | Beat should play | Plays | Y |
| 110 | Set dial sample 7 beat 4 | Beat should play | Plays | Y |
| 111 | Set dial sample 7 beat 5 | Beat should play | Plays | Y |
| 112 | Set dial sample 7 beat 6 | Beat should play | Plays | Y |
| 113 | Set dial sample 7 beat 7 | Beat should play | Plays | Y |
| 114 | Set dial sample 7 beat 8 | Beat should play | Plays | Y |
| 115 | Set dial sample 7 beat 9 | Beat should play | Plays | Y |
| 116 | Set dial sample 7 beat 10 | Beat should play | Plays | Y |
| 117 | Set dial sample 7 beat 11 | Beat should play | Plays | Y |
| 118 | Set dial sample 7 beat 12 | Beat should play | Plays | Y |
| 119 | Set dial sample 7 beat 13 | Beat should play | Plays | Y |
| 120 | Set dial sample 7 beat 14 | Beat should play | Plays | Y |
| 121 | Set dial sample 7 beat 15 | Beat should play | Plays | Y |
| 122 | Set dial sample 7 beat 16 | Beat should play | Plays | Y |
| 123 | Set dial sample 8 beat 1 | Beat should play | Plays | Y |
| 124 | Set dial sample 8 beat 2 | Beat should play | Plays | Y |
| 125 | Set dial sample 8 beat 3 | Beat should play | Plays | Y |
| 126 | Set dial sample 8 beat 4 | Beat should play | Plays | Y |
| 127 | Set dial sample 8 beat 5 | Beat should play | Plays | Y |
| 128 | Set dial sample 8 beat 6 | Beat should play | Plays | Y |
| 129 | Set dial sample 8 beat 7 | Beat should play | Plays | Y |
| 130 | Set dial sample 8 beat 8 | Beat should play | Plays | Y |
| 131 | Set dial sample 8 beat 9 | Beat should play | Plays | Y |
| 132 | Set dial sample 8 beat 10 | Beat should play | Plays | Y |
| 133 | Set dial sample 8 beat 11 | Beat should play | Plays | Y |
| 134 | Set dial sample 8 beat 12 | Beat should play | Plays | Y |
| 135 | Set dial sample 8 beat 13 | Beat should play | Plays | Y |
| 136 | Set dial sample 8 beat 14 | Beat should play | Plays | Y |
| 137 | Set dial sample 8 beat 15 | Beat should play | Plays | Y |
| 138 | Set dial sample 8 beat 16 | Beat should play | Plays | Y |
| 139 | Set dial sample 9 beat 1 | Beat should play | Plays | Y |
| 140 | Set dial sample 9 beat 2 | Beat should play | Plays | Y |
| 141 | Set dial sample 9 beat 3 | Beat should play | Plays | Y |
| 142 | Set dial sample 9 beat 4 | Beat should play | Plays | Y |
| 143 | Set dial sample 9 beat 5 | Beat should play | Plays | Y |
| 144 | Set dial sample 9 beat 6 | Beat should play | Plays | Y |
| 145 | Set dial sample 9 beat 7 | Beat should play | Plays | Y |
| 146 | Set dial sample 9 beat 8 | Beat should play | Plays | Y |
| 147 | Set dial sample 9 beat 9 | Beat should play | Plays | Y |
| 148 | Set dial sample 9 beat 10 | Beat should play | Plays | Y |
| 149 | Set dial sample 9 beat 11 | Beat should play | Plays | Y |
| 150 | Set dial sample 9 beat 12 | Beat should play | Plays | Y |
| 151 | Set dial sample 9 beat 13 | Beat should play | Plays | Y |
| 152 | Set dial sample 9 beat 14 | Beat should play | Plays | Y |
| 153 | Set dial sample 9 beat 15 | Beat should play | Plays | Y |
| 154 | Set dial sample 9 beat 16 | Beat should play | Plays | Y |
| 155 | Set dial sample 10 beat 1 | Beat should play | Plays | Y |
| 156 | Set dial sample 10 beat 2 | Beat should play | Plays | Y |
| 157 | Set dial sample 10 beat 3 | Beat should play | Plays | Y |
| 158 | Set dial sample 10 beat 4 | Beat should play | Plays | Y |
| 159 | Set dial sample 10 beat 5 | Beat should play | Plays | Y |
| 160 | Set dial sample 10 beat 6 | Beat should play | Plays | Y |
| 161 | Set dial sample 10 beat 7 | Beat should play | Plays | Y |
| 162 | Set dial sample 10 beat 8 | Beat should play | Plays | Y |
| 163 | Set dial sample 10 beat 9 | Beat should play | Plays | Y |
| 164 | Set dial sample 10 beat 10 | Beat should play | Plays | Y |
| 165 | Set dial sample 10 beat 11 | Beat should play | Plays | Y |
| 166 | Set dial sample 10 beat 12 | Beat should play | Plays | Y |
| 167 | Set dial sample 10 beat 13 | Beat should play | Plays | Y |
| 168 | Set dial sample 10 beat 14 | Beat should play | Plays | Y |
| 169 | Set dial sample 10 beat 15 | Beat should play | Plays | Y |
| 170 | Set dial sample 10 beat 16 | Beat should play | Plays | Y |
| 171 | Set dial sample 11 beat 1 | Beat should play | Plays | Y |
| 172 | Set dial sample 11 beat 2 | Beat should play | Plays | Y |
| 173 | Set dial sample 11 beat 3 | Beat should play | Plays | Y |
| 174 | Set dial sample 11 beat 4 | Beat should play | Plays | Y |
| 175 | Set dial sample 11 beat 5 | Beat should play | Plays | Y |
| 176 | Set dial sample 11 beat 6 | Beat should play | Plays | Y |
| 177 | Set dial sample 11 beat 7 | Beat should play | Plays | Y |
| 178 | Set dial sample 11 beat 8 | Beat should play | Plays | Y |
| 179 | Set dial sample 11 beat 9 | Beat should play | Plays | Y |
| 180 | Set dial sample 11 beat 10 | Beat should play | Plays | Y |
| 181 | Set dial sample 11 beat 11 | Beat should play | Plays | Y |
| 182 | Set dial sample 11 beat 12 | Beat should play | Plays | Y |
| 183 | Set dial sample 11 beat 13 | Beat should play | Plays | Y |
| 184 | Set dial sample 11 beat 14 | Beat should play | Plays | Y |
| 185 | Set dial sample 11 beat 15 | Beat should play | Plays | Y |
| 186 | Set dial sample 11 beat 16 | Beat should play | Plays | Y |
| 187 | Set dial sample 12 beat 1 | Beat should play | Plays | Y |
| 188 | Set dial sample 12 beat 2 | Beat should play | Plays | Y |
| 189 | Set dial sample 12 beat 3 | Beat should play | Plays | Y |
| 190 | Set dial sample 12 beat 4 | Beat should play | Plays | Y |
| 191 | Set dial sample 12 beat 5 | Beat should play | Plays | Y |
| 192 | Set dial sample 12 beat 6 | Beat should play | Plays | Y |
| 193 | Set dial sample 12 beat 7 | Beat should play | Plays | Y |
| 194 | Set dial sample 12 beat 8 | Beat should play | Plays | Y |
| 195 | Set dial sample 12 beat 9 | Beat should play | Plays | Y |
| 196 | Set dial sample 12 beat 10 | Beat should play | Plays | Y |
| 197 | Set dial sample 12 beat 11 | Beat should play | Plays | Y |
| 198 | Set dial sample 12 beat 12 | Beat should play | Plays | Y |
| 199 | Set dial sample 12 beat 13 | Beat should play | Plays | Y |
| 200 | Set dial sample 12 beat 14 | Beat should play | Plays | Y |
| 201 | Set dial sample 12 beat 15 | Beat should play | Plays | Y |
| 202 | Set dial sample 12 beat 16 | Beat should play | Plays | Y |
| 203 | Set dial sample 13 beat 1 | Beat should play | Plays | Y |
| 204 | Set dial sample 13 beat 2 | Beat should play | Plays | Y |
| 205 | Set dial sample 13 beat 3 | Beat should play | Plays | Y |
| 206 | Set dial sample 13 beat 4 | Beat should play | Plays | Y |
| 207 | Set dial sample 13 beat 5 | Beat should play | Plays | Y |
| 208 | Set dial sample 13 beat 6 | Beat should play | Plays | Y |
| 209 | Set dial sample 13 beat 7 | Beat should play | Plays | Y |
| 210 | Set dial sample 13 beat 8 | Beat should play | Plays | Y |
| 211 | Set dial sample 13 beat 9 | Beat should play | Plays | Y |
| 212 | Set dial sample 13 beat 10 | Beat should play | Plays | Y |
| 213 | Set dial sample 13 beat 11 | Beat should play | Plays | Y |
| 214 | Set dial sample 13 beat 12 | Beat should play | Plays | Y |
| 215 | Set dial sample 13 beat 13 | Beat should play | Plays | Y |
| 216 | Set dial sample 13 beat 14 | Beat should play | Plays | Y |
| 217 | Set dial sample 13 beat 15 | Beat should play | Plays | Y |
| 218 | Set dial sample 13 beat 16 | Beat should play | Plays | Y |
| 219 | Set dial sample 14 beat 1 | Beat should play | Plays | Y |
| 220 | Set dial sample 14 beat 2 | Beat should play | Plays | Y |
| 221 | Set dial sample 14 beat 3 | Beat should play | Plays | Y |
| 222 | Set dial sample 14 beat 4 | Beat should play | Plays | Y |
| 223 | Set dial sample 14 beat 5 | Beat should play | Plays | Y |
| 224 | Set dial sample 14 beat 6 | Beat should play | Plays | Y |
| 225 | Set dial sample 14 beat 7 | Beat should play | Plays | Y |
| 226 | Set dial sample 14 beat 8 | Beat should play | Plays | Y |
| 227 | Set dial sample 14 beat 9 | Beat should play | Plays | Y |
| 228 | Set dial sample 14 beat 10 | Beat should play | Plays | Y |
| 229 | Set dial sample 14 beat 11 | Beat should play | Plays | Y |
| 230 | Set dial sample 14 beat 12 | Beat should play | Plays | Y |
| 231 | Set dial sample 14 beat 13 | Beat should play | Plays | Y |
| 232 | Set dial sample 14 beat 14 | Beat should play | Plays | Y |
| 233 | Set dial sample 14 beat 15 | Beat should play | Plays | Y |
| 234 | Set dial sample 14 beat 16 | Beat should play | Plays | Y |
| 235 | Set dial sample 15 beat 1 | Beat should play | Plays | Y |
| 236 | Set dial sample 15 beat 2 | Beat should play | Plays | Y |
| 237 | Set dial sample 15 beat 3 | Beat should play | Plays | Y |
| 238 | Set dial sample 15 beat 4 | Beat should play | Plays | Y |
| 239 | Set dial sample 15 beat 5 | Beat should play | Plays | Y |
| 240 | Set dial sample 15 beat 6 | Beat should play | Plays | Y |
| 241 | Set dial sample 15 beat 7 | Beat should play | Plays | Y |
| 242 | Set dial sample 15 beat 8 | Beat should play | Plays | Y |
| 243 | Set dial sample 15 beat 9 | Beat should play | Plays | Y |
| 244 | Set dial sample 15 beat 10 | Beat should play | Plays | Y |
| 245 | Set dial sample 15 beat 11 | Beat should play | Plays | Y |
| 246 | Set dial sample 15 beat 12 | Beat should play | Plays | Y |
| 247 | Set dial sample 15 beat 13 | Beat should play | Plays | Y |
| 248 | Set dial sample 15 beat 14 | Beat should play | Plays | Y |
| 249 | Set dial sample 15 beat 15 | Beat should play | Plays | Y |
| 250 | Set dial sample 16 beat 16 | Beat should play | Plays | Y |
| 251 | Set dial sample 16 beat 1 | Beat should play | Plays | Y |
| 252 | Set dial sample 16 beat 2 | Beat should play | Plays | Y |
| 253 | Set dial sample 16 beat 3 | Beat should play | Plays | Y |
| 254 | Set dial sample 16 beat 4 | Beat should play | Plays | Y |
| 255 | Set dial sample 16 beat 5 | Beat should play | Plays | Y |
| 256 | Set dial sample 16 beat 6 | Beat should play | Plays | Y |
| 257 | Set dial sample 16 beat 7 | Beat should play | Plays | Y |
| 258 | Set dial sample 16 beat 8 | Beat should play | Plays | Y |
| 259 | Set dial sample 16 beat 9 | Beat should play | Plays | Y |
| 260 | Set dial sample 16 beat 10 | Beat should play | Plays | Y |
| 261 | Set dial sample 16 beat 11 | Beat should play | Plays | Y |
| 262 | Set dial sample 16 beat 12 | Beat should play | Plays | Y |
| 263 | Set dial sample 16 beat 13 | Beat should play | Plays | Y |
| 264 | Set dial sample 16 beat 14 | Beat should play | Plays | Y |
| 265 | Set dial sample 16 beat 15 | Beat should play | Plays | Y |
| 266 | Set dial sample 16 beat 16 | Beat should play | Plays | Y |

## Software Listings

QT Open Source SDK – 2009.05 version

DirectX SDK 2010 February version

RTAudio 4.0.7

STK 4.4.2