Plymouth University

School of Computing, Electronics and Mathematics

PRCO304

Final Stage Computing Project

2016/2017

BSc (Hons) <*Computer Science*>

<*Mark Anthony Powell*>

<*10452876*>

<*Automatic Music Notation Maker*>

Supervisor: <Thomas Wenneker>

Second marker: <your second marker’s name>

Table of Contents

[Chapter 1 – Introduction 4](#_Toc486867649)

[1.1 Summary 4](#_Toc486867650)

[1.2 The Project Aim 4](#_Toc486867651)

[1.3 Motivation 4](#_Toc486867652)

[1.3 Objectives 5](#_Toc486867653)

[1.5 Minimum Requirements 5](#_Toc486867654)

[1.6 Possible Enhancements 6](#_Toc486867655)

[Chapter 2 – Planning 6](#_Toc486867656)

[2.1 Planning 7](#_Toc486867657)

[2.2 Supervisor Meetings 8](#_Toc486867658)

[Chapter 3 – BAckGround Research 8](#_Toc486867659)

[3.1 Introduction 8](#_Toc486867660)

[3.2 Music Notation 8](#_Toc486867661)

[3.2.1 Guitar Tablature 8](#_Toc486867662)

[3.2.2 Sheet Music 9](#_Toc486867663)

[3.3 Digital Audio Representation 11](#_Toc486867664)

[3.3.1 Noise Sampling 11](#_Toc486867665)

[3.3.2 Audio Buffers 11](#_Toc486867666)

[3.4 Similar Problems 11](#_Toc486867667)

[3.2.1 Mindfusion Scorewriter 12](#_Toc486867668)

[3.2.2 Auto-Tune 12](#_Toc486867669)

[3.2.3 WPF MIDI Band 12](#_Toc486867670)

[Chapter 4 – The solution 13](#_Toc486867671)

[4.1 Libraries Used 17](#_Toc486867672)

[4.2 The Design 18](#_Toc486867673)

[4.2.1 Why C# 18](#_Toc486867674)

[4.2.1 The Class Structure 18](#_Toc486867675)

[4.3 The Visualization and Interface 18](#_Toc486867676)

[Chapter 5 – Evaluation 18](#_Toc486867677)

[5.1 Self Evaluation 18](#_Toc486867678)

[5.1.1 18](#_Toc486867679)

[5.2 Users evaluation 18](#_Toc486867680)

[Chapter 6 – References 18](#_Toc486867681)

[[1] How to Read Sheet Music: Step-by-Step Instructions 18](#_Toc486867682)

[[2] Sibelius - the leading music composition and notation software 18](#_Toc486867683)

[[3] Sibelius | Customer Stories | Avid 18](#_Toc486867684)

[[4] NAudio 19](#_Toc486867685)

[[5] How does a computer record and produce sound? 19](#_Toc486867686)

[[6] Realtime C&#35; Pitch Tracker 19](#_Toc486867687)

[[7] Pitch detection algorithm 19](#_Toc486867688)

[[8] Autotune.NET 19](#_Toc486867689)

[[9] Automatic Pitch Detection and Shifting of Musical Tones in Real Time 20](#_Toc486867690)

[[10] LINE LEVEL, INSTRUMENT LEVEL, MIC LEVEL EXPLAINED 20](#_Toc486867691)

[[10] HOW TO CONVERT BYTE[] TO SHORT[] OR FLOAT[] ARRAYS IN C# 20](#_Toc486867692)

# Chapter 1 – Introduction

## Summary

Whether it be rock, pop, classical, jazz or electro, everyone has an interest in music. With some, that interest is keen enough to entice listeners to learn how to create their own compositions or lean how to play songs composed by others. The frequency of the tones comprising a composition are represented in an easy to understand format known as “notes”. Given a string of notes, a musician is instructed in how to play that melody. This report will provide an understanding as to how a received sound can be interpreted by a computer and represented visually in a form musicians may understand.

## The Project Aim

The aim of the project is to ease the music creating experience of musicians by allowing the consisting musical notes to be dictated with little additional effort to the musician. To this end, software needed to be created which would recognize the notes that are being played as well as draw a graphical representation of what is being played. The graphical representation is to be produced as the musician is playing the notes (in real-time).

## 1.3 Motivation

Any final year project is without exception, prove to be extremely difficult. To provide a descent end product requires steady hard work for an extended period of time. It is for this reason that I felt it necessary to choose a project theme that I am personally enthusiastic about in order to keep motivation going. My interest in music made this the prime subject for my final year project. My intent was to produce a final piece of software that I would enjoy using myself.

For most, the ability/will to create music comes in sudden bursts of fragile inspiration. An idea for a tune ignites like a spark but goes out and is forgotten just as easily if it is not written down. To do so however, may cause the musician to lose their train of thought and cause the burst of inspiration to fade. This project inspires to allow the musician to create music, free of diversions and to keep easy track of riffs they have previously composed.

## Objectives

The objectives for this project were as follows: -

\*To research and study the various methods and techniques used in audio signal processing.

\*To implement an application capable of recognizing a string of musical notes as they are played and present them graphically.

\*To produce an interface which corresponds with Schneiderian golden rules of usability.

\* To assess the challenges which presented themselves during the construction of this project and conclude as to how these issues may be mitigated in future.

## 1.5 Minimum Requirements

The minimum requirements as discussed with the project supervisor.

\*To detect the pitch produced from an audio source such as a microphone, direct line input etc.

\*To identify the notes of the relevant identified pitches.

\*To apply those notes to a visual representation a musician may easily interpret.

\*To provide an interface which will enable to user to start the guitar notation process.

## 1.6 Possible Enhancements

\*To implement additional features such as allowing the recording and playback of music.

\* To provide the ability to save and read rotation to a .txt file.

\* To allow MIDI playback of created notation.

\*To recognize and facilitate the use of chords as well as single notes.

# Chapter 2 – Planning

## 2.1 Planning

To carry out this project, it became necessary for me to acquire greatly detailed knowledge about concepts I had previously had only minimal knowledge concerning. It was necessary to understand the way in which a computer can represent the same sounds we hear with our ears in to binary numbers.

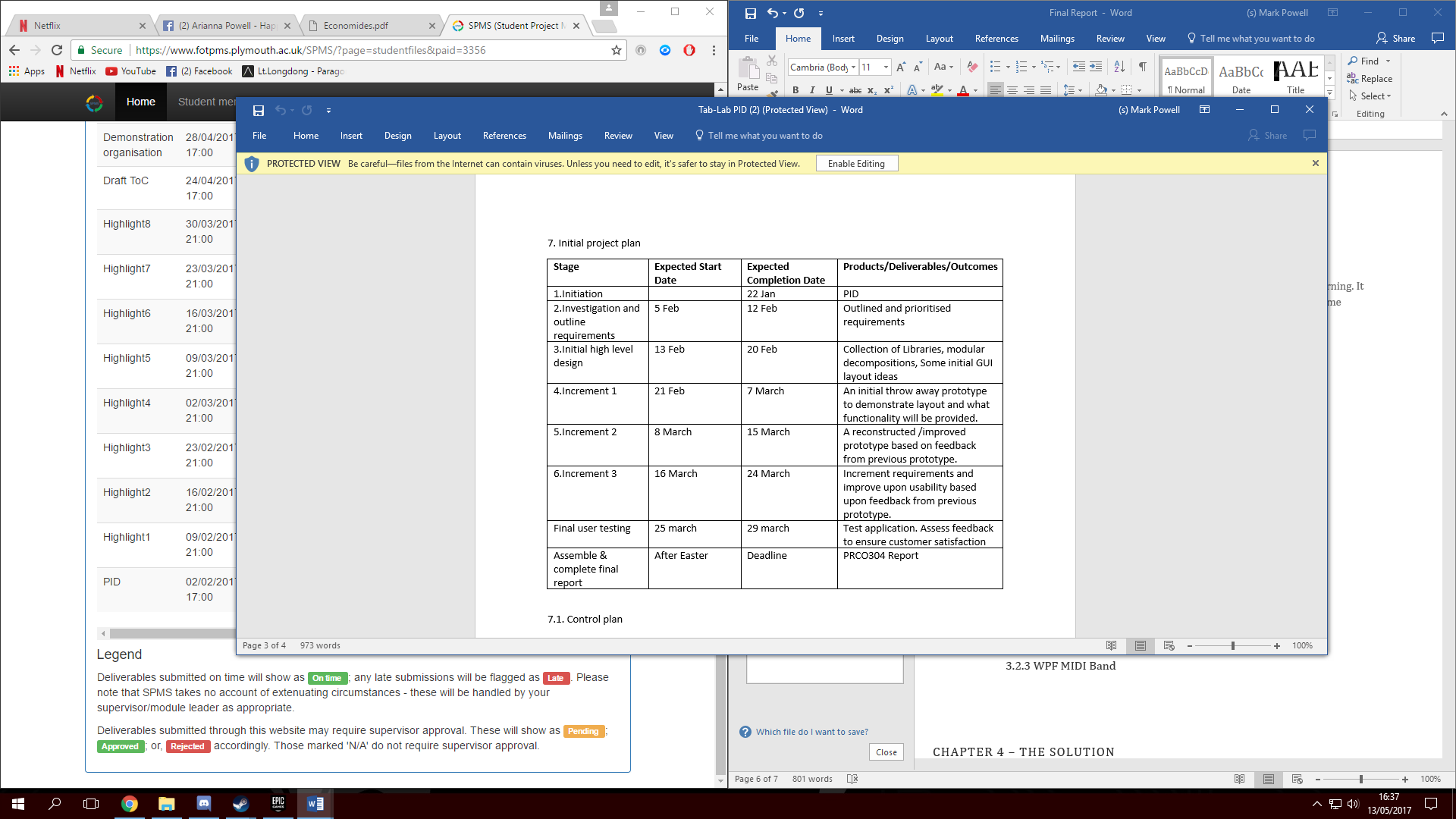


Figure 1 - original project plan from PID

The original project plan was based around the intention to quickly build a working prototype with the basic functionality. From there, I was to carry on improving upon its usability, adding additional features as each stage of the prototype underwent testing. It failed to comprehend the actual complexity of the main functionality.

It ended up taking far longer to detect a pitch than had been expected. For a while, I was unable to get the Pitchtracker library to work and attempted to create my own class for implementing a fast Fourier transform technique to detect the pitch. Trying to understand and implement this FFT technique proved to be difficult and time consuming. I was eventually able to get the pitch tracker library to work however this diversion caused a big nock on effect with my project schedule.

## 2.2 Supervisor Meetings

To ensure the project was on track, weekly meetings were set up with my assigned supervisor. These meetings were to allow me to elaborate and talk through the contents provided within the progress reports. My supervisor would then help clarify the progress is should have made at this point and provide me with a new milestone goal to accomplish in time for our next meeting.

# Chapter 3 – BAckGround Research

## 3.1 Introduction

To this point, the modules covered over the years at university provided the foundation of knowledge necessary to code and solve problems on our own. To meet the standards expected of a final year project requires many hours of additional learning and practice. My project is music based and as such requires a rather in depth knowledge in how music is visually represented as well as recognized by a machine. It was necessary to understand these concepts in order to pick apart the problem and identify the steps that needed it be implemented. Some basic knowledge of the terminology used is also useful in understanding how I have structured my solution.

## 3.2 Music Notation

### 3.2.1 Guitar Tablature

There are various ways and themes in which music notation may take visual form. Many of these forms however finely suit a specified instrument for which it is to be played on. The notation form I am most familiar with is known as “tablature”. Tablature is a very simple and easy to use visual representation designed for the use of plucked string instruments such as guitar, bass or ukulele. In the case of guitars, there are 6 lines to represent the 6 strings. Time flows from left to right where numbers are placed along the lines to represent the fret that should be played. The reason this method is easy to use and understand is that the representation bears close resemblance to the way in which the instrument is being played. It requires little understanding behind musical theory or even much explanation in how to read.

For these reasons, this was the method of representation I had originally planned upon using, hence the original project name “tab-Lab”. This representation method would have however, proved problematic for multiple reasons. when notating based on a received pitch. The same frequencies/notes can be achieved by holding corresponding frets on different strings. This makes it difficult to identify which fret is being played and could provide messy, unorganized results of closely sounding notes jumping across the fretboard.

### 3.2.2 Sheet Music

Although I play an instrument myself, guitar songs are commonly presented in the form of “tablature” as opposed to sheet music. Until recently, the Guitar was the only instrument I played and so my knowledge of sheet music was very limited. I have recently started to learn how to play piano which uses sheet music. However, I had only covered the basics at the point of starting this project. This project is aimed to facilitate various instruments and so required a more universal representation than tablature was able to provide. For this purpose, sheet music is the ideal form of representation as its symbols represent which note is to be played as opposed to properties specific to an instrument (e.g. string, fret, key etc.)

Sheet music or “score”, is a written or printed form of notation used to represent a musical melody. It contains all the properties and information required to play a melody such as notes, the length of each note, the rhythm etc. Written music can go in to detail as to the precise length of each note and even when to pause and for how long. This project however, does not go in to such detail and therefore this report will only clarify those factors relevant to the solution. [1]

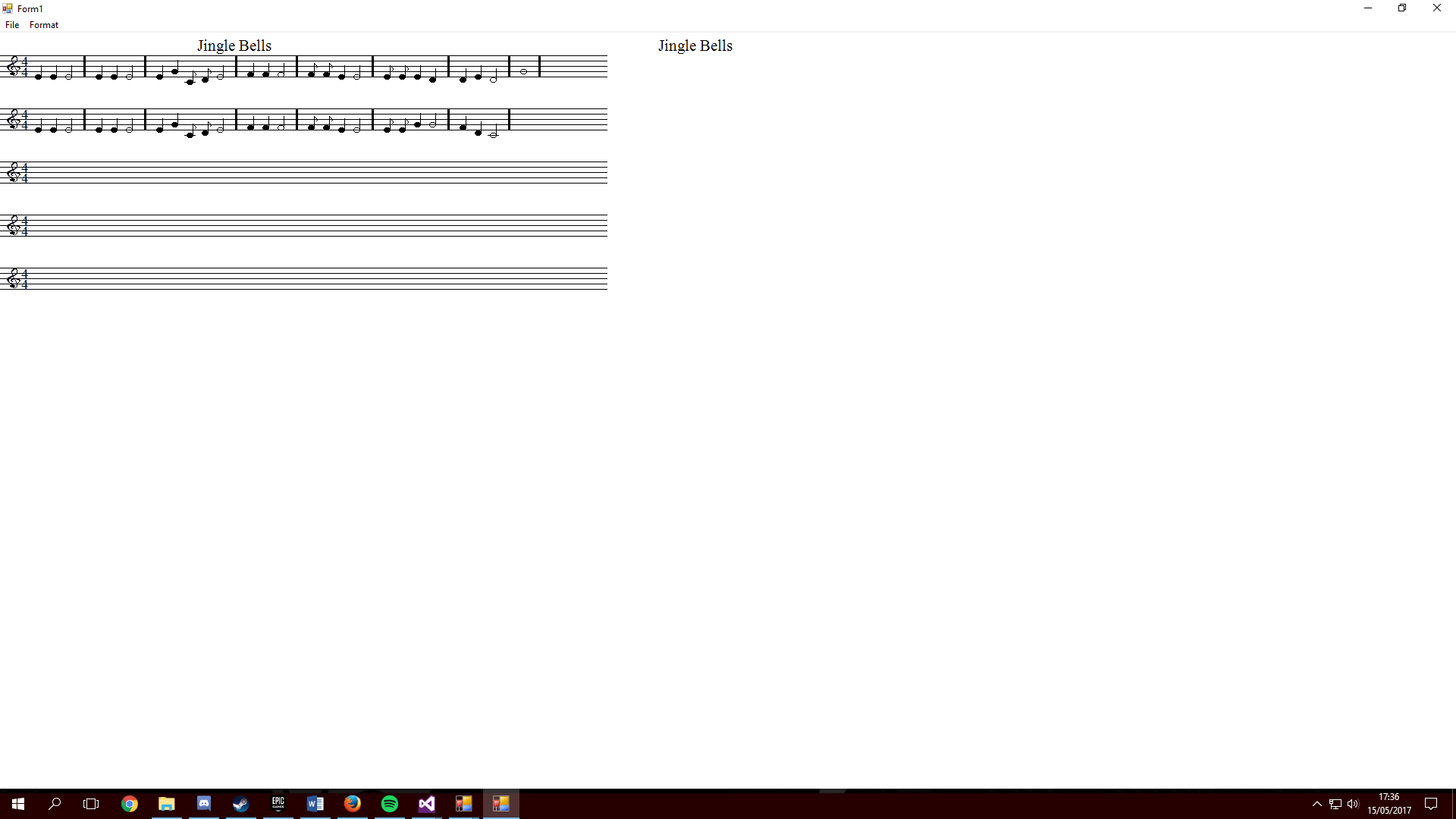


Figure 2 - An example of the song “jingle Bells” presented via sheet music.

#### 3.2.1 Staff & Notes

The 5 lines represent a note alphabetically in the order of E, G, B, D, F and the spaces in between represent the other F, A, C, E notes (see figure 3).



Figure 3 - Shows which note each line represents

An octave consists of 12 alphabetically named notes. These notes go in the order of A, A#, B ,C ,C# ,D ,D# ,E ,F ,F# ,G and G# . Any sound higher or lower than this simply work their way back through the order but become part of a different octave. My solution allows for three different octaves (low, middle and high) out of ten that are given in the musical range. If a note falls in to the lower octave, the note is placed below the staff with a line through it to help identify its position. Similarly, a note that falls in to the higher octave is placed above.

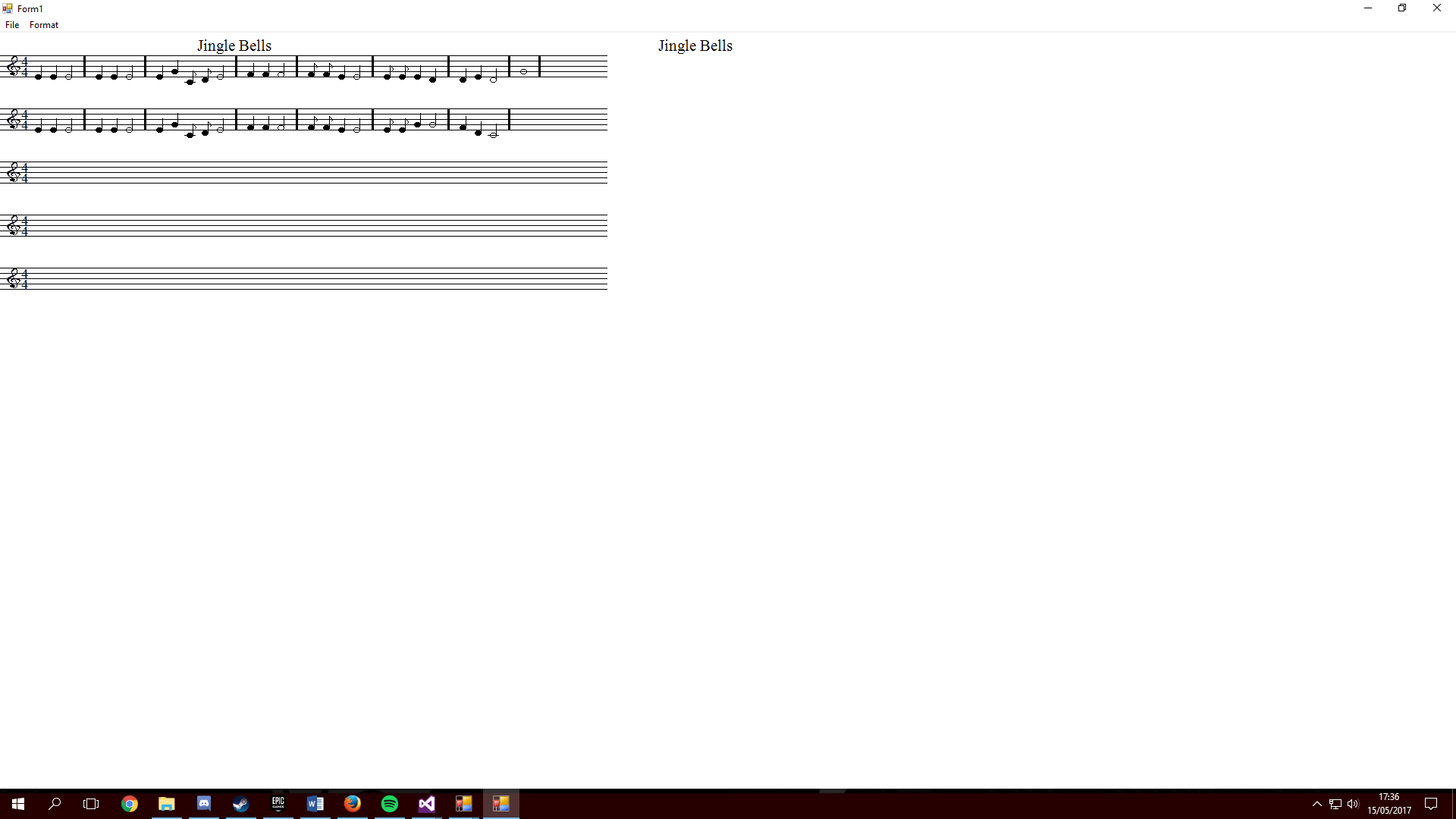


Figure 4 - Shows a note falling outside the staff boundaries

Each note contains a range of frequencies that correspond with that note

#### 3.2.2 Measures and signatures

Sheet music is broken up into small time segments called “measures” or “bars”. These measures are divided by a line such as seen in figure 5.

The numbers to the left of a staff are used to identify the beat and timing of a melody. These numbers are called the songs “time Signature”. The top number identifies how many beats are in each measure whereas the bottom number gives a value to each note per beat. For example, 4/4 means there are 4 beats per bar and every quarter note equals one beat.

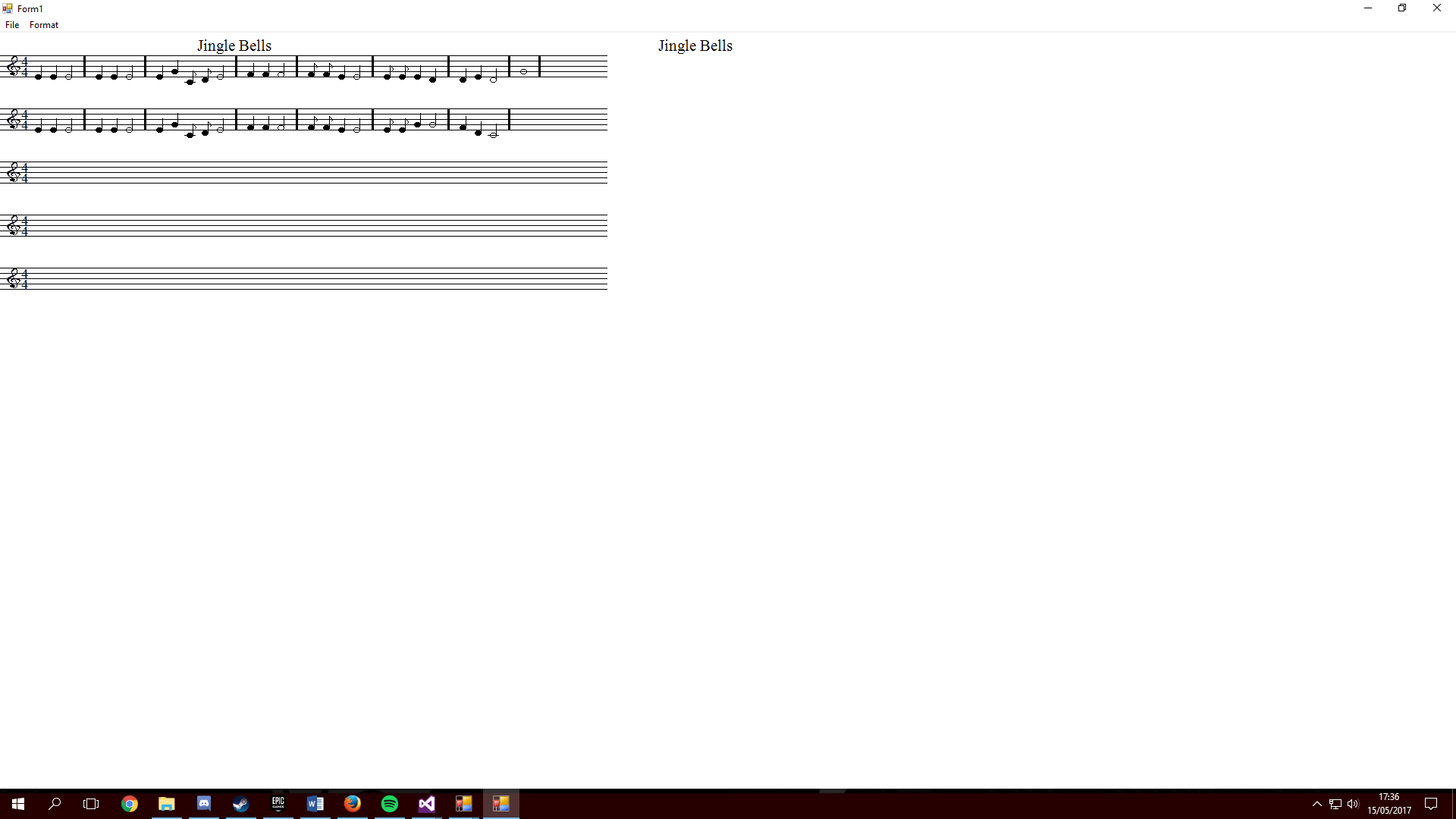


Figure 5 - Shows the use of signatures and measures in sheet music

#### 3.2.3 Clefs

There are two clefs in sheet music; The treble clef and the bass clef. The bass clef corresponds to lower pitched notes or chords played alongside the rest of the song. This project only on the main part of the song currently however and so only the treble clef is used.

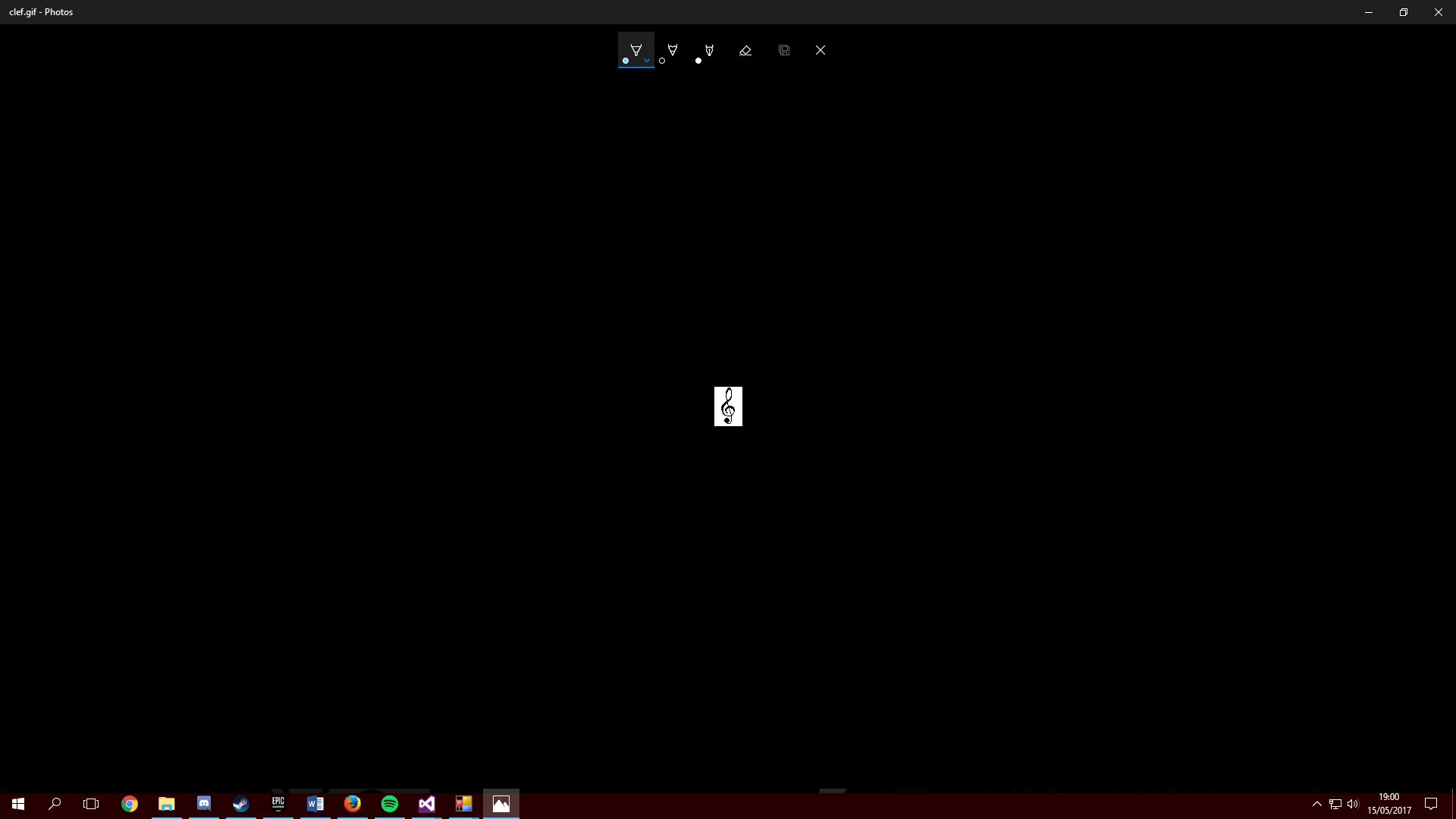


Figure 6 - The treble clef drawn in the project

## 3.3 Digital Audio Representation

### 3.3.1 Noise Sampling

### 3.3.2 Audio Buffers

## 3.4 Similar Problems

There are many applications that allow complicated and quite accurate sheet music to be provided from analyzing a pre-recorded song. “Sibelius”[2] for example, is a software tool provided by Avid which allows the user to create and edit sheet music to a high level of detail.

Sibelius can work from either pre-recorded tracks or allows for virtual instrument input in real-time. They call this feature “Flexitime” and it allows for notation to be produced based on input from plug in MIDI instruments or the virtual instrument interface. The tool is suitable for use by professionals or learners and has been used in composing music for blockbuster films such as James bond and by sound designers for companies like Dolby [3]. The makers of the application focused on the interface in making it as intuitive and versatile to edit as possible. The application allows MIDI playback of compositions and even the ability to convert them in to MP3 files for transfer. It allows multiple instruments simultaneously, provides detailed timing indications and is capable of recognizing chords as well as notes.

Providing sheet music from real instruments or singing in real-time however is much more complicated and far fewer applications exist that provide this functionality. Some that exist however can work with a high degree of accuracy and usability. After emailing Sibelius technical support to request more detail on their flexi-time input, they pointed me in the direction of “Audioscore”, for a product that provides such functionality [12].

“Audioscore” is either usable on its own or allows its results to be sent directly to programs such as Sibelius for a better editing experience. As well as the note, Audioscore is also capable of identifying volume changes, timing and even identifying the instrument that is being played. It can even be used to provide feedback as to the musician’s accuracy when trying to play the given sheet music.

I was unable to locate any documentation detailing the pitch detection algorithm used in the Sibelius’s real-time input tool and so emailed to request further information. Their technical support team informed me that their real-time input relies on the use of MIDI instruments

the implementation of the software on their website or on the internet. As such, I decided to email the company support team to request they point me towards any such documentation they may have on the subject.

Neuratron for example have created their own tool for creating scores by singing or playing in to a microphone. The tool can be used on its own or entwined with popular sheet music editing software such as “Sibelius”.

### 3.2.1 Mindfusion Scorewriter

### 3.2.2 Auto-Tune

Autotune.NET is an example program created by the creator of NAudio, Mark Heath. It was created as a means of demonstrating the capabilities of the NAudio library and how it can be used to implement real-time pitch detection. Auto-tuning is a technique used by singers to ensure they’re singing is always on key. Auto-Tuning software does this by detecting the current pitch and shifting it to the correct one. The example program from NAudio isn’t very fine tuned and the results are a little robotic sounding. The program however proved useful for identifying the fundamentals behind pitch detection and better understand how buffers work.

### 3.2.3 WPF MIDI Band

# Chapter 4 – The solution

The solution to this problem can be broken in to three separate processes.

1: To receive audio and translate it in to data.

2: To detect the pitch and therefore note values

3: To Transcribe those note values in to sheet music.

When broken down like this, it is easy to be led to believe that the solution is simple. In reality however, nothing involving audio processing, pitch detection or using code to draw comes even close to simple. Thankfully, program libraries already exist that apply their own methods to ensure the data is as accurate and easier to work with.

To receive audio and translate it in to data.

Like most things, computers use the laws of physics to understand and interpret audio in to a language they are capable of understanding. Sound is caused by tiny fluctuations traveling through the air. These fluctuations create wave forms which a transducer (such as a microphone) is capable of recording and relaying to the computer [5]. These waveforms are usually intensely erratic and complicated; yet must be converted in to digital data for the computer to process.

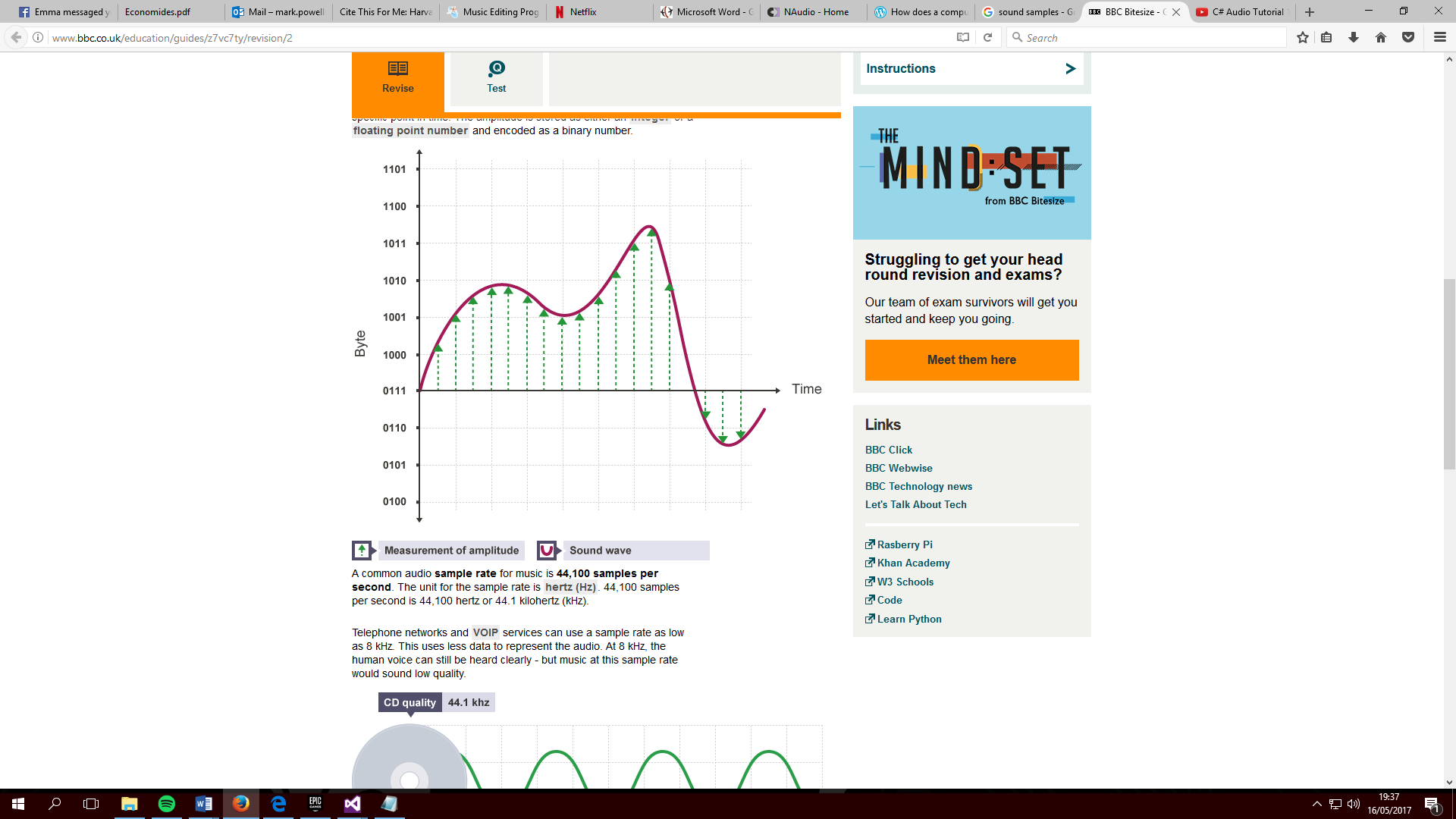


Figure 7 - How samples are taken from an audio wave form - http://www.bbc.co.uk/education/guides/z7vc7ty/revision/2

A computer transfers analogue waveforms in to digital data by sampling the analogue waveform extremely quickly and return sound values for those sample points.

NAudio [4] is an open sourced audio and MIDI library to assist development of audio related utilities. The solution uses the NAudio library to record the wave data and transfer it to a buffer with values we can work with. Naudio also provides functionality which allows us to save and playback recorded audio, set up the input device and to see the waveforms visually.

To help do this, NAudio contains methods which allow the user to easily change convert a byte audio buffer in to a short of floating point buffer. [10] This is a task very common in audio coding as it allows each sample to be accessed directly. For each byte of data received from the soundcard, the value needs to be converted in to samples in bits equal to that of the recording quality (e.g 16 bit or 32 bit). Although NAudio contains methods that do this process for you, I have chosen to convert the types manually as to better see what is happening. As you can see from figure 8, each byte of audio data received is added to the buffer. Those bytes are converted in to standard 16bit short samples which is then converted in to 32bit float samples and added to the floatbuffer. That buffer is periodically processed using the Pitchtracker library upon every time the timer ticks.

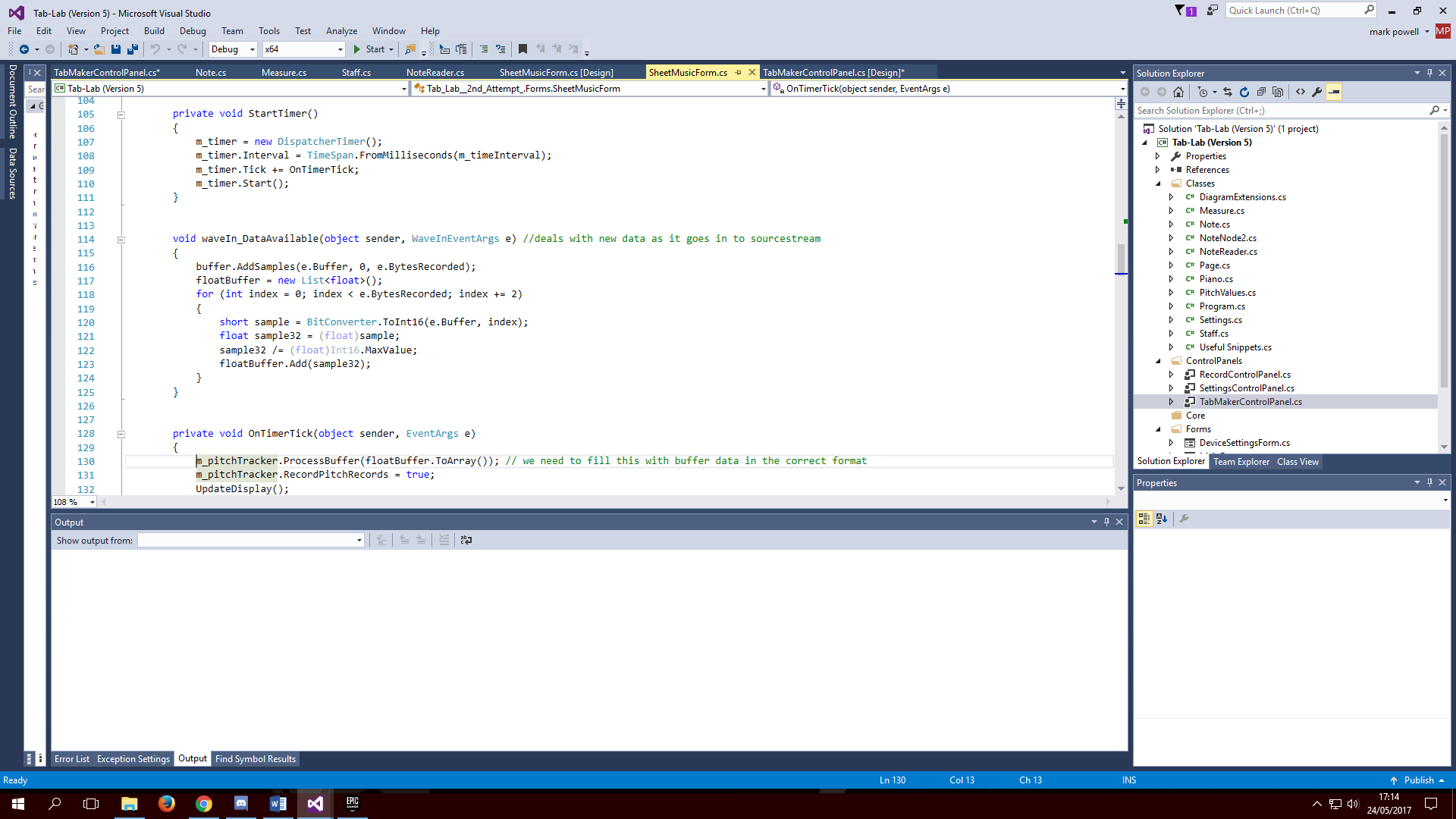


Figure 8

**To detect the pitch and therefore note values**

A very complex science goes in to the methods used to calculate pitch and even then, accuracy is not guaranteed. Calculating pitch becomes increasingly more difficult when dealing with multiple pitches(polyphonic) at once such as chords or multiple instruments simultaneously. It is for that reason that this project is restricted to working with only one instrument performing single notes(monophonic).

There are many different techniques used to detect pitch. Each technique is better geared towards its own purpose such as voice recognition or digital recording. Pitch detection algorithms typically fall in to one of the two main categories, “time domain” and “frequency domain” however there are those that hybrid between the two.

Fast Fourier Transform

The most commonly known method is known as the “Fast Fourier Transform” technique. This technique converts signals from the “time Domain”, in to the “frequency domain”, allowing it to be efficient as well as accurate. This allows it to be used in more complex applications such as polyphonic pitch detection and auto-voice tuning. I had looked in to using this algorithm. I had even constructed a tutorial auto-voice tuning application using NAudio to better understand how it works and how to use it [8].

Auto-Correlation

[9] This method falls under the time domain category. Autocorrelation computes the similarity between its original signal and a time lagged version of that same signal. From here, it can identify the dominant pitch by finding the fundamental frequency. The issue being that this comparison needs to be checked with every sample. This quickly becomes a slow process when dealing with high resolution audio due to the high sampling rate involved.

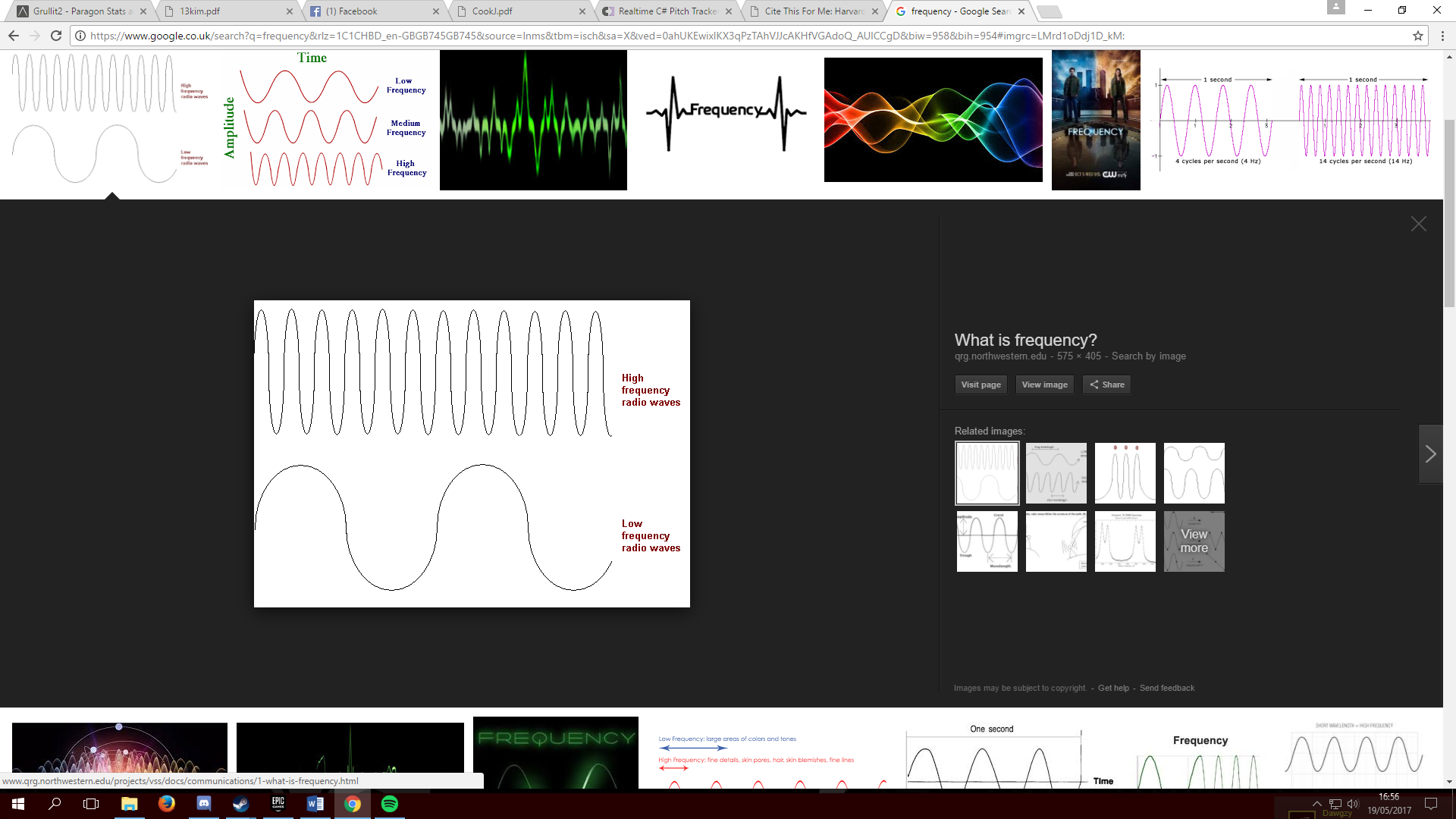


Figure 9 - demonstrates the effect of frequency in sound waves

[6] The standard way in which auto-correlation works also causes lower accuracy as frequencies become higher. As you can see in figure 8, high frequencies contain less time in between cycles. Less time allows for less samples per cycle. Window size stepping is given in whole sample values causing changes in frequency steps to become broader as the frequency becomes higher.

The Pitch tracker library used in the solution, utilizes a heavily modified version of the auto-correlation algorithm which is designed to iron out the flaws found in normal auto-correlation. One such modification, is using the ratios of different peaks instead of absolute values. This allows pitch testing to remain accurate even at higher frequency levels with results only measured to have deviated from the actual frequency by a tolerance of 0.02%. This is more than accurate enough for identifying a note.

As well as frequency, another factor that may potentially effect accuracy when using auto-correlation is the strength of the audio signal. That is, that the accuracy can be effected by the power of the audio device being used. In this case, that could either be a microphone or a guitar lead. [10] The rough measurements of a direct instrument output such as a guitar usually stand around -20 dBu and standard quality microphones around -30dBu. Pitchtracker claims to remain accurate between the ranges of -40dB to 0dB, which means there should be no issue with any input device we are likely to use.

Despite being based on the auto-correlation algorithm, pitchtracker is still able to maintain high testing speeds of around 3000 pitch tests per second. This is achieved by allowing two passes. The first pass only checks every 8th sample whilst using course steps in frequency. The second phase is then able to provide higher resolution testing focused only on the frequency detected in the first pass. This is much faster than the normal way auto-correlation performs comparisons on every sample.

**To Transcribe those note values in to sheet music**

As explained earlier, sheet music portrays much more about a song than simply which notes are played. If a string of notes were to be played at a constant speed, one after another, the song would likely be unrecognizable to hear. A lot of what makes up a song lays in the beat and tempo. This greatly added to the complexity of the project as there are a vast set of complex rules which differed based on the previously defined measurements. However, it was felt that without taking timing in to consideration, the transcription wouldn’t provide a very good indication of how the song was to be played.

## 4.1 Libraries Used

### 4.2 The Design

### 4.2.1 Why C#

Due to the complexity of this project, it was recommended that I not build the entire thing from scratch and instead try to utilize already existing libraries relevant to the project. For this reason, compatibility with my preferred libraries was a main determining factor in choosing which language I would be using. Part of determining the feasibility of this project was identifying which libraries I would be using. C# appeared to be the most common language used in audio related libraries. Once I had found a library for capturing audio input and detecting its pitch, it made sense to abide by the same language they both co-existed within.

Another language I had considered was java. I had previously implemented a user interface using netbeans in constructing our project for our “user-Centred Interface Deisgn” module in second year. I had worked intensely hard on the project and got very good at customizing my forms to make them look just right. I believe netbeans provides better customization options than visual studio for producing the layout of a Graphical interface.

### 4.2.1 The Class Structure

### 4.3 The Visualization and Interface

### 4.3.1 The Theme

Ive made attempts to add some cosmetics in to the application using design software provided for free by the university such as “expression Studio” and photoshop. The mistake was that I created skins for the buttons and artistic titles for each dynamic panel early on in the project. Afterwards, I changed my mind as to the functionality of each panel and therefore the title would also need to change.

Many features of my applications interface are similar to those commonly used in most apps. Shneidermans 8 golden rules of interface design states that the designer should “strive for consistency”[13]. This consistency refers to consistency within the application such as terminology used or location of buttons with similar functions. It also refers to consistency with the way the user would expect the application to work based on how other applications work. This required me to take a look at some other applications and see what common characteristics emerge.

The Tab Panel – The boldest and defined of functionality (e.g Recording studio and the composer), are navigated to via large, bold buttons on the left hand side panel. This is the common style for many applications and therefore easy to locate.

There is currently plenty of space for me to add additional functionality as I see fit.

# Chapter 5 – Evaluation

## 5.1 Self Evaluation

### 5.1.1

## 5.2 Users evaluation

# Chapter 6 – References

### [1] How to Read Sheet Music: Step-by-Step Instructions

**In-text:** (Musicnotes.com Blog, 2017)

**Your Bibliography:** Musicnotes.com Blog. (2017). *How to Read Sheet Music: Step-by-Step Instructions*. [online] Available at: http://www.musicnotes.com/blog/2014/04/11/how-to-read-sheet-music/ [Accessed 15 May 2017].

### [2] Sibelius - the leading music composition and notation software

**In-text:** (Sibelius.com, 2017)

**Your Bibliography:** Sibelius.com. (2017). *Sibelius - the leading music composition and notation software*. [online] Available at: http://www.sibelius.com/products/audioscore/ultimate.html [Accessed 16 May 2017].

### [3] Sibelius | Customer Stories | Avid

**In-text:** (Avid.com, 2017)

**Your Bibliography:** Avid.com. (2017). *Sibelius | Customer Stories | Avid*. [online] Available at: http://www.avid.com/en/sibelius/customer-stories [Accessed 16 May 2017].

### [4] NAudio

**In-text:** (CodePlex, 2017)

**Your Bibliography:** CodePlex. (2017). *NAudio*. [online] Available at: https://naudio.codeplex.com/ [Accessed 16 May 2017].

### [5] How does a computer record and produce sound?

**In-text:** (Music and Computer Science, 2017)

**Your Bibliography:** Music and Computer Science. (2017). *How does a computer record and produce sound?*. [online] Available at: https://musicandcomputerscience.wordpress.com/how-does-a-computer-record-and-produce-sound/ [Accessed 16 May 2017].

### [6] Realtime C&#35; Pitch Tracker

**In-text:** (CodePlex, 2017)

**Your Bibliography:** CodePlex. (2017). *Realtime C&#35; Pitch Tracker*. [online] Available at: https://pitchtracker.codeplex.com/documentation [Accessed 17 May 2017].

### [7] Pitch detection algorithm

**In-text:** (En.wikipedia.org, 2017)

**Your Bibliography:** En.wikipedia.org. (2017). *Pitch detection algorithm*. [online] Available at: https://en.wikipedia.org/wiki/Pitch\_detection\_algorithm [Accessed 17 May 2017].

### [8] Autotune.NET

**In-text:** (Channel 9, 2017)

**Your Bibliography:** Channel 9. (2017). *Autotune.NET*. [online] Available at: https://channel9.msdn.com/coding4fun/articles/AutotuneNET [Accessed 17 May 2017].

### [9] Automatic Pitch Detection and Shifting of Musical Tones in Real Time

**In-text:**(Kim, 2017)

**Your Bibliography:**Kim, J. (2017). *Automatic Pitch Detection and Shifting of Musical Tones in Real Time*. 1st ed. Boston, pp.16-24. http://cslab1.bc.edu/~csacademics/pdf/13kim.pdf

### [10] LINE LEVEL, INSTRUMENT LEVEL, MIC LEVEL EXPLAINED

**In-text:**(Ovnilab.com, 2017)

**Your Bibliography:**Ovnilab.com. (2017). *Line level, instrument level, mic level explained*. [online] Available at: http://www.ovnilab.com/articles/linelevel.shtml [Accessed 20 May 2017].

### [11] HOW TO CONVERT BYTE[] TO SHORT[] OR FLOAT[] ARRAYS IN C#

**In-text:**(Mark-dot-net.blogspot.co.uk, 2017)

**Your Bibliography:**Mark-dot-net.blogspot.co.uk. (2017). *How to convert byte[] to short[] or float[] arrays in C#*. [online] Available at: http://mark-dot-net.blogspot.co.uk/2013/03/how-to-convert-byte-to-short-or-float.html [Accessed 24 May 2017].

### [12] TRANSCRIBE MP3S & CD TRACKS WITH AUDIOSCORE

**In-text:**(Neuratron.com, 2017)

**Your Bibliography:**Neuratron.com. (2017). *Transcribe MP3s & CD Tracks with AudioScore*. [online] Available at: http://www.neuratron.com/audioscore.htm [Accessed 5 Jul. 2017].

### [13] SHNEIDERMAN’S EIGHT GOLDEN RULES WILL HELP YOU DESIGN BETTER INTERFACES

**In-text:**(The Interaction Design Foundation, 2017)

**Your Bibliography:**The Interaction Design Foundation. (2017). *Shneiderman’s Eight Golden Rules Will Help You Design Better Interfaces*. [online] Available at: https://www.interaction-design.org/literature/article/shneiderman-s-eight-golden-rules-will-help-you-design-better-interfaces [Accessed 12 Jul. 2017].