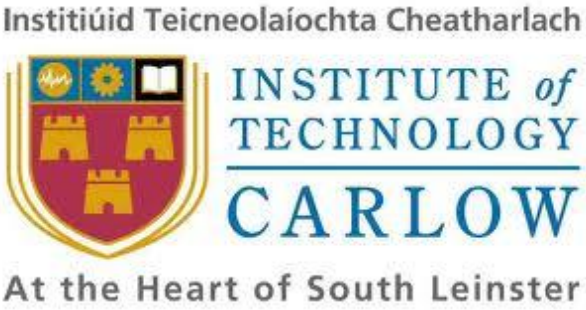

Project Report

Institute of Technology, Carlow

B.Sc.(Honours) in Software Development

CW238

Project Title

Musical Tablature Editor

Document Date

17/04/13

Student Name and ID

Robert Connolly C00123951

Supervisor

Paul Barry

Table of Contents

Introduction

This document reviews and concludes the activities on my finished project. It will describe:

- What the finished product is.
- How the finished product compares to the specification and original design.
- What did I technically and personally learn and achieve.
- Finally, give a complete review of my project in terms of what experience I had with it and what advice I have to give to anyone else attempting a similar project.

The aim of this project was to develop an application that allows users to write down or create their music on the electric, acoustic and bass guitar and drums in a very simple way. That way is tablature. It was aimed at non musicians and musicians who don't know musical notation. This would be aided with the ability to hear what their music sounds like, by being able to play back what they've written. This allows users to hear if they have written down their music accurately, but also help them create music by being able to hear what they've written sounds like. The project was also to aid others in learning another person's music, by being able to read it in the application and play back the music to hear what it sounds like.

Description of Submitted Project

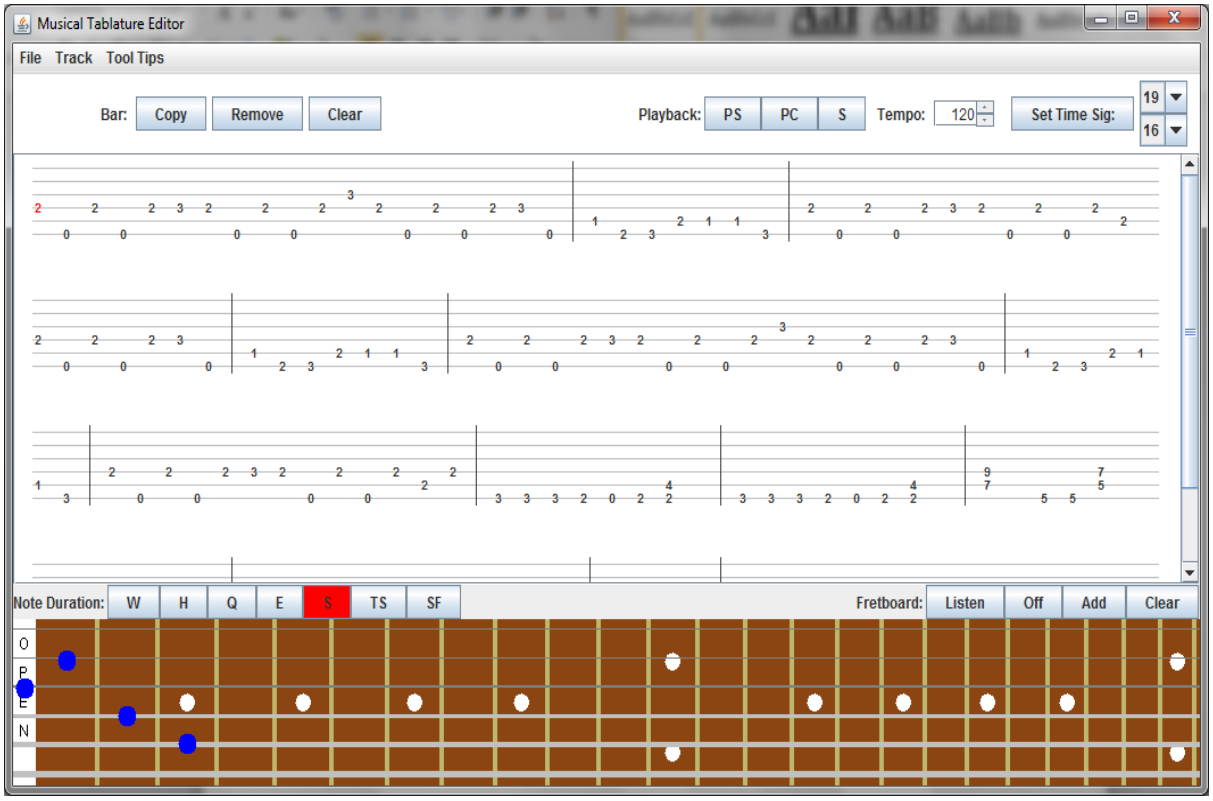

Fig 1: Screenshot of the finished product.

The finished product allows users to write down their music for a six string electric or acoustic guitar in tablature form and play it back. I will describe it and what I've achieved in terms of the core features of my project and extra functionalities, which are:

1. **Electric/Acoustic Guitar Tablature**
2. **Note Effects and Durations**
3. **MIDI Playback**
4. **Extra Features**

1. Electric/Acoustic Guitar Tablature

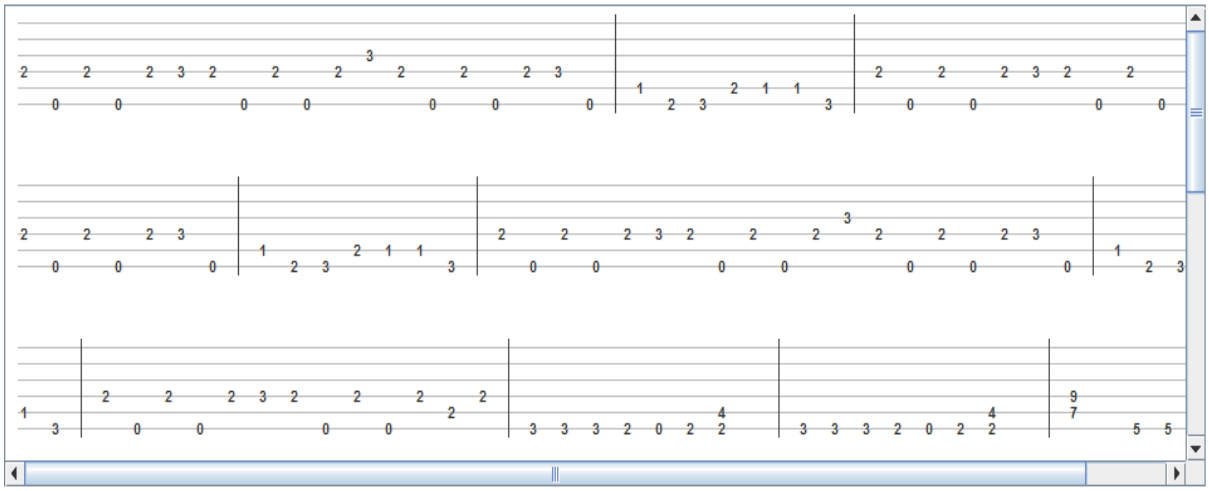

Fig 2: Music written in tablature in the finished product.

The user can write all the tablature they need for their music. *Fig 2 s*hows how it is represented in the application. It is an exact replication of how tablature is written. When the tablature gets too big for the window as the user writes, scrollbars appear to allow the user to move with ease through their tablature, see *Fig 2.*

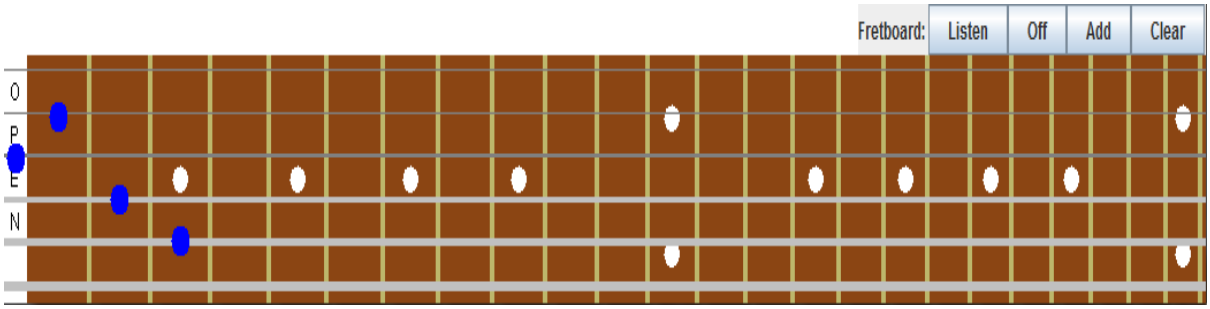

Fig 3: Using the fretboard at the bottom of the screen to write tablature.

The tablature in *Fig 2* is written using the fretboard in *Fig 3.* It is an exact replication of how a fretboard looks on a guitar. It resizes to fit the window to whatever size is comfortable for the user.

To write on to the tablature, the user selects notes with the mouse on the fretboard and then can add them to the tablature. The blue dots in *Fig 3* show what notes the user has selected at this point in time. The users selection is achieved by invisible buttons placed on top of the drawn fretboard. They line up exactly with the drawn frets that can be seen on the fretboard, and resize exactly when the user changes the size of the window.

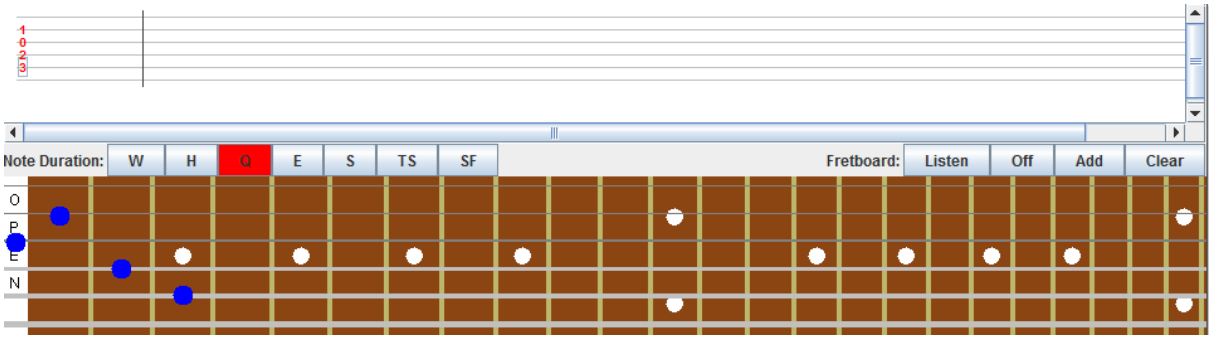

Fig 4: Selecting notes on the tablature to update.

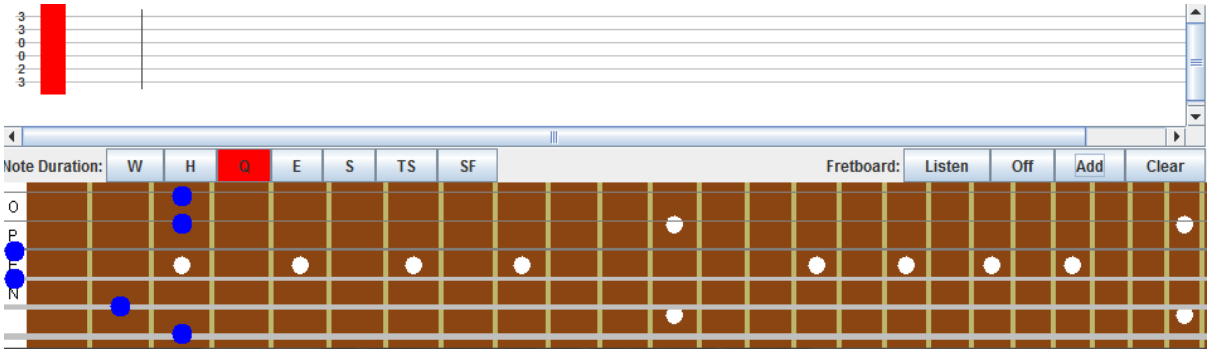

Fig 5: Notes updated on the tablature.

The user can update any notes in the tablature by simply selecting a note on the tablature. The note and any notes played with it appear on the fretboard, and they can be edited as the user wishes, see *Fig 4*. When they add the notes back to the tablature, they replace exactly what they selected on the fretboard, see *Fig 5*.

They can delete individual notes on the fretboard, or clear the whole fretboard and start again.

Other features of the fretboard are that they can hear what the notes sound like together as they add and delete them from the fretboard. This feature can be turned off. They can also hear what they've added to the fretboard at any time by pressing a button. All this allows the user to hear what they're adding to the tablature before they add. It helps to make sure what they are adding is accurate. Also, they can create interesting chords this way by being able to experiment and hear what they are like.

Fig 6: Default look of toolbar for manipulating bars in the music.

The components shown in *Fig 6*, allow the user to copy, remove and clear bars of music written in the tablature. These operations apply to one bar at a time.

Copying bars allows the user to insert bars quickly into the tablature that repeat in the music. This happens quite often in music, so it would be very tedious if the user had to rewrite identical bars in the music all of the time.

If the user is unhappy with a bar in the music, they can simply remove it from the tablature. This is crucial removing misplaced bars and to update rewritten music.

The user can also clear any bar. This leaves the bar in the tablature but clears all the notes in it. This allows them to rewrite a bar very quickly if they made a mess of it.

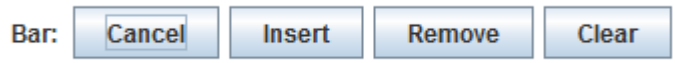

Fig 7: What Fig 6 looks like when the user copies a bar in the tablature.

Fig 7 is what *Fig 6* looks like when the user copies a bar. It allows them to insert a copied bar as many times as they like in the music simply by pressing the 'Insert' button repeatedly. It aids greatly in enhancing the experience of writing their tablature if they can add in repeated bars very quickly. It removes the tedium of writing repeated music.

Fig 8: This sets the time signature of a particular bar.

Time signatures, see *Fig 8*, are a very important component in writing bars. They allow you to set how many beats are in a bar, and what's the duration of those beats.

The top value in the time signature is how many beats are in the bar. The bottom value represents the time duration or value of each of the beats. This is not to say all the notes in the bar will have the time duration designated by the time signature. It is more about defining how long overall the bar is in duration. Therefore, this allows the user to define exactly how long the bar is in duration. As a result, they can set the bar to perfectly fit the exact amount notes of whatever note durations they need.

The need for this is because it is crucial in writing the music accurately. The bars will only be as long as they should exactly be, and with only the notes that should be in them. This means the music is in perfect sync rhythmitically. Music is all about timing, so this feature is crucial.

2. Note Effects and Durations

The note effects were not implemented in the finished product, so I will not describe them in this document.

Fig 9: Toolbar to change note durations.

Fig 9 shows the toolbar for changing a note's duration. Notes in music always have different durations at some point within the music. This toolbar allows the user to change the duration of any note. This is another crucial feature of the finished product, as the user can write how long each note is to be played for. This also allows the MIDI to playback the user's tablature accurately with the notes playing for the correct duration.

Overall, being able to change the note durations is crucial in writing music to tablature and learning the music accurately.

3. MIDI Playback

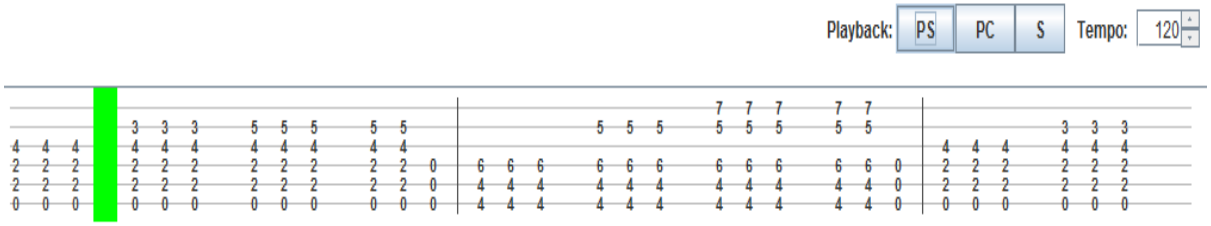

Fig 10: Playing back the tablature in the finished product.

This toolbar is a core feature allows the user to play back what tablature they have written, using MIDI. See *Fig 10* for the playback toolbar. MIDI tells the sound device the instructions of how to play the music, such as when to play notes and for how long. Using the note durations the user has applied to their notes, the MIDI will play back the users tablature exactly as written.

This feature is so important in allowing the user to know exactly how the music sounds. Therefore, it's crucial for writing their existing music accurately in the tablature and for others to be able to learn the music accurately, as they can hear exactly how it should be played.

Also when the user wants to create new music, they can hear exactly what it sounds like. This is a pivotal aid in creating music.

The user can play the tablature from start to finish, or from any point in the tablature to the finish. They can also stop playback at any time.

During playback, the tablature is highlighted in green, see *Fig 10,* to show the user where they are in the song and what's being exactly played at that point in time.

In *Fig 10,* a tempo tool can be seen. This sets the speed at which the music is played back at. When writing existing music to tablature or creating new music, this allows the user to tell if the music is being played back at the right tempo and change it if not. It also allows anyone learning another's music, at what tempo to play it at.

4. Extra Features

a. New/Open/Save tablature.
b. Choosing an Instrument.
c. Choosing a Key.
d. Tooltips

a. New/Open/Save tablature

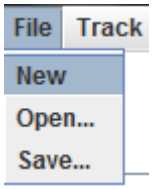

Fig 11: The file menu in the finished product.

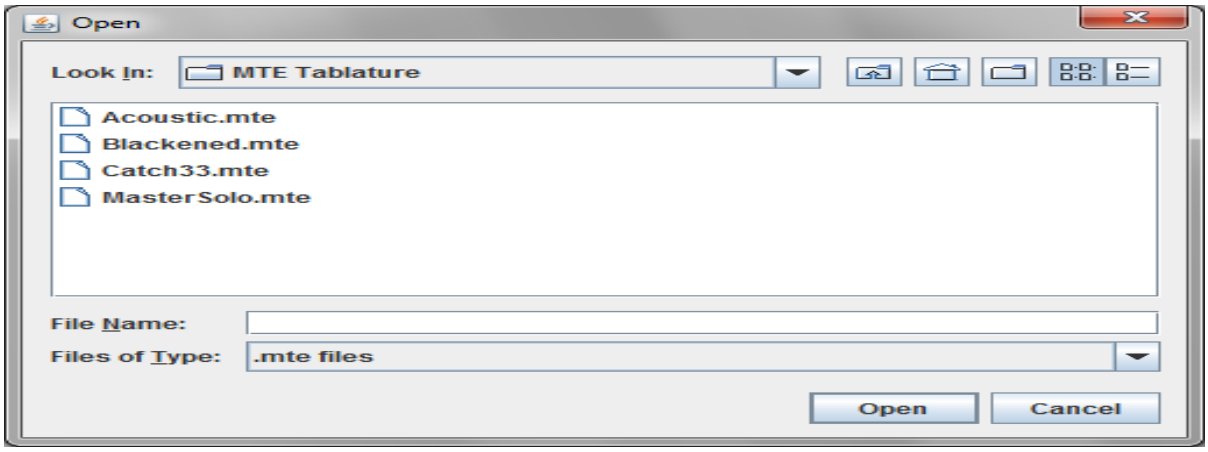

Fig 12: Opening an existing tablature.

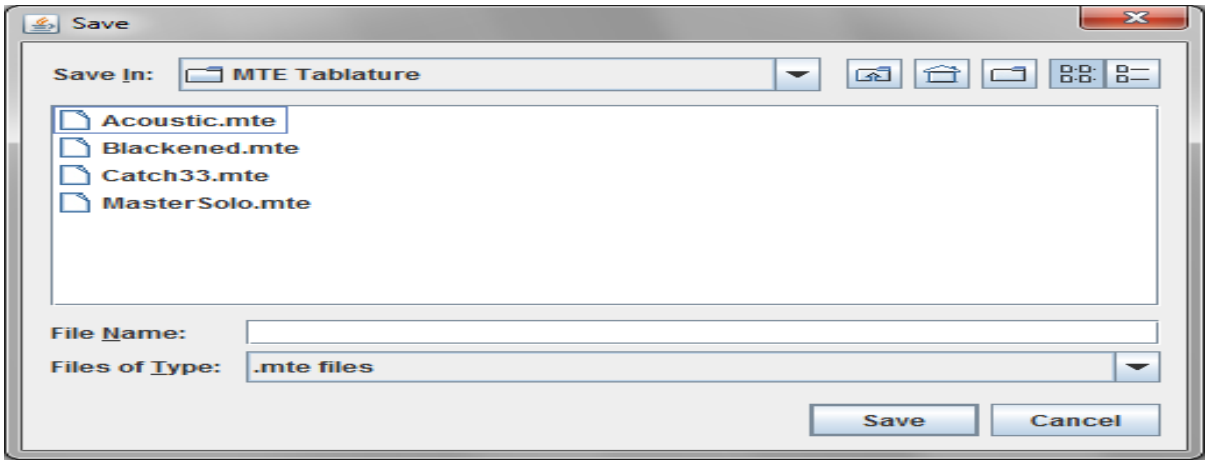

Fig 13: Saving the currently written tablature.

The user can create a new tablature to write at any time(*Fig 11)*, open a previously saved tablature(*Fig 12)*, or save one they are working on(*Fig 13)*.

The user needs to be able to save the tablature they are writing, otherwise the product lacks any longevity. They would not be able to document their music, or be able to return to it if they didn't have enough time to write it first time around. Other users would have no access to learning other user's music either. So, while it's a small feature, it is a very important and powerful one.

Also, the product reads what the user's home directory is and automatically creates a folder, called "*MTE Tablature*", especially for them to save their work, see *Fig 12 and 13.* They can, however, save tablature in any folder they wish.

b. Choosing an Instrument

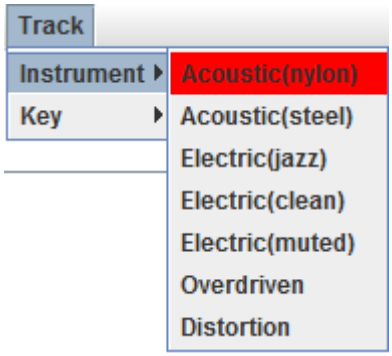

Fig 14: Choosing an instrument in the finished product.

The user can choose whatever instrument they like from the list in *Fig 14.* This is for use in the MIDI play back of their tablature. When playing back the tablature, it's of an enormous aid to be able to hear what the music sounds like when played the target instrument to play it.

It's a far more immersive experience. The user can create music better when the instrument they are using is what they hear when they play back the tablature. They can write their existing music with more tonal accuracy, and others can learn the music with more accuracy, because they can hear what way the music sounds with the correct instrument. It adds depth and a more enjoyable experience to the users.

The instrument in use will be highlighted at all times, so the user knows what they are using and can change it if they so wish.

c. Choosing a Key

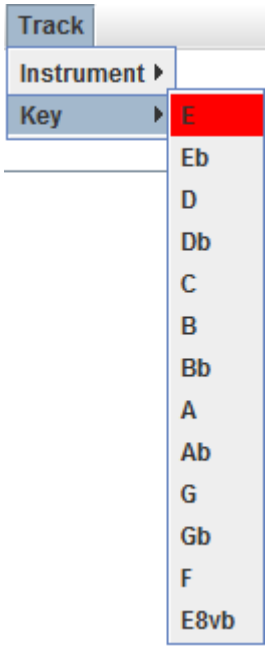

Fig 15: Choosing a music key in the finished product.

The user can select whatever music key they want their music to be played in. The MIDI plays back their music in the exact key chosen.

Music played in different keys can have a profound effect on the quality of the music overall. What sounds good in one key, may not sound good in another. Different keys create different feeling and tone. The user can select the exact key they want their music played back in. It is crucial in accurately portraying the music in exactly how it sounds. It aids the creative process greatly as they can experiment writing new music in different keys.

Usually they will know what their key is, so straight away they can select it from the menu and be ready to go. Users learning the music can hear exactly what the music sounds like and in what key, so they can get the correct feel for it.

All keys that are generally used in music are available to the user. So, they'll never have to worry about their music being portrayed in the wrong key when playing it back.

d. ToolTips

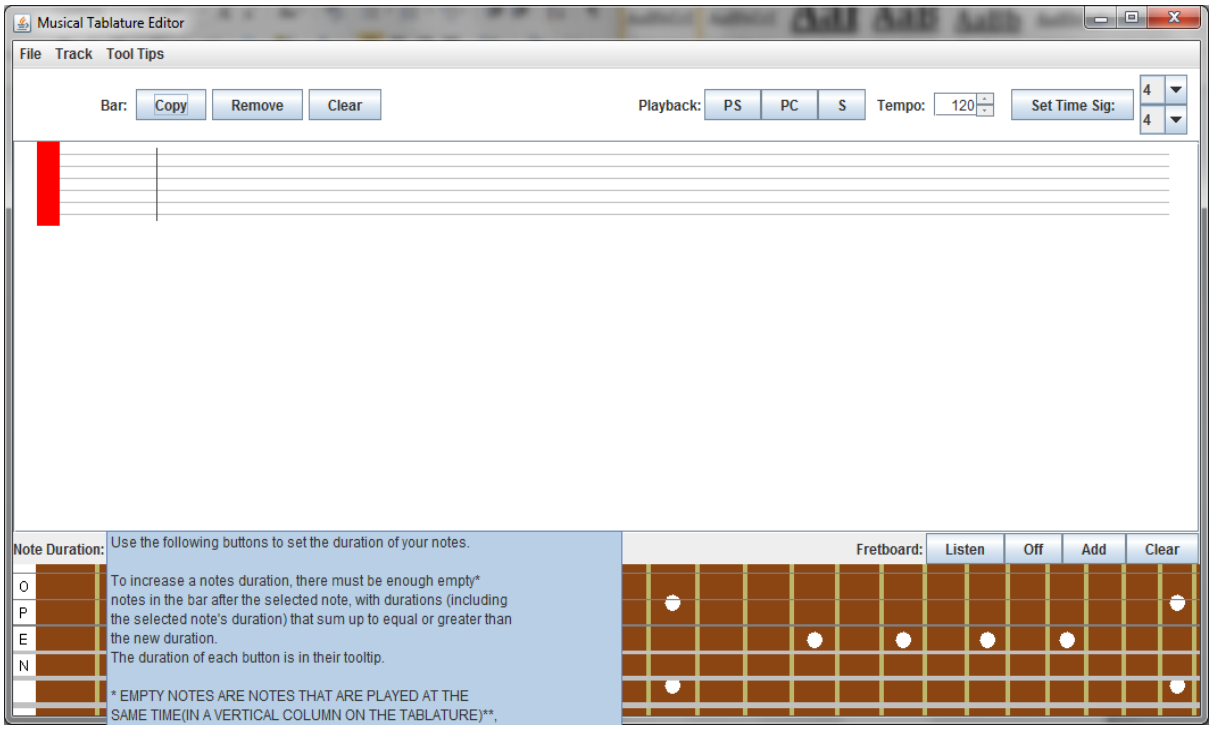

Fig 16: Tooltips in the finished product.

The user can learn how to use every feature in the application simply by putting their mouse cursor over any of them. The user can turn this feature on and off.

Conformance to Specification and Design

Due to time constraints of my honours degree course, the only realistic target was to be able to implement what were considered my core features. I will discuss conformance to specification and design in terms of those core features and data structures used:

1. **Electric/Acoustic Guitar Tablature**
2. **Note Effects and Durations**
3. **MIDI Playback**
4. **Classes and Data Structures**

1. Electric/Acoustic Guitar Tablature

Specification:

This feature conforms exactly to the specification. There is a mention of a design decision which should not be in the specification. This will be dealt with in the conformation of the design on page 13*.

Design:

- **Add Notes**

The design of adding notes to the tablature has changed. The way I had it before was, that note objects were created as the user adds new notes. The user wouldn't know if they could add the notes to the bar or with the duration they needed, because that would only be checked when they tried to add the notes. Also, the notes could get a different duration than the user intended, depending on what space was left in the bar. This most likely would not be the duration the user intended. It would become very messy. The user would not really know exactly what space was left in the bar to add notes, and what was going on with any possible note duration changes.

Now, the bar they will be entering notes into, will have already been created. It will be filled with empty notes objects of a certain duration, depending on the time signature of the previous bar.

I have done this because it lets the user know the exact size of the bar and what notes objects are in it, and what durations each notes object has. The user can change the time signature or any note durations with full confidence of what they are doing. They know what notes are in the bar at all times, and know what notes they can add or change in a bar.

- **Delete Notes**

Before, the design was that the user deleted the notes and the notes were removed from the bar.

What happens now is, the user simply overwrites notes by adding from the fretboard when it is empty. I've done this because it maintains the time signature of the bar. It still contains all the notes of durations that add up to duration of the whole bar(which is determined by the signature).

*One difference was for when the user wants to enter the next chord or note to play, the user presses the right arrow key to move to the next part. What happens instead is, when the user adds notes to the tablature, the next insertion point for notes on the tablature is automatically selected for them.

This is because it allows a quicker insertion of notes that are repeated one after another. All the user has to do is keep pressing the 'Add' button on the fretboard, and identical notes are inserted one after another very quickly. The fretboard maintains what notes the user initially entered for this to work. It is meant to help writing tablature easier and quicker.

2. Note Effects and Durations

I did not get to develop and implement the note effects, so I will just concentrate on note durations here.

Specification:

The note durations conform exactly to the specification.

Design:

The note durations largely conform to the design. The differences are:

- Instead of messaging the user if a note duration change is too large to change it, a system sound is heard indicating it can't be done. I have done this because that is largely consistent with how operations that can't be done are messaged to the user on operating systems. For example, trying to access a window that opened the current window in Windows 7. The sound is familiar to users as a result.

- In the design before, the bar itself was checked for space to see if it could accommodate the note's change in duration. Instead, the notes objects in the bar are now being checked. The duration of the note being changed and the duration of empty notes objects in the bar after it, are considered space. Not the bar's space itself, as previously designed.
  If the change is longer than the current duration, the note to be changed and any empty notes in the bar after it are considered empty space. This is used to

calculate if the note's duration can be changed to longer. Any empty notes objects allowing for this are removed.
Also, if the note duration change is shorter than the current one, the current duration of the note is considered as space. As a result, notes objects of the new duration will be created and inserted in the bar to make up for the old duration.

The reason for this is because the bar has already been created and is filled with notes objects, and as a result has no space left. The notes objects themselves are the potential space.

3. MIDI Playback

Specification:

The MIDI playback conforms exactly to the specification.

Design:

The design is identical, except in one regard. The duration of the note to be played is not calculated into microseconds, as per design. It is calculated in MIDI ticks.

The beat the note stops playing on, is based on its durations in MIDI ticks also. Not the index in which the note sits in the array list in the bar, as per my design.

The reason for this is because that's how the Java Sound measures the duration of a note to be played using MIDI.

4. Classes and Data Structures

The classes and the data structures used conform exactly to my design.

The only differences are the renaming of some attributes to more descriptive names, and the addition of other attributes as needed.

The data type for tracks in the tablature was changed to an interface called 'Tracks'.

This is because it was intended to have drum tracks as well as guitar tracks. Both would have a generic type to implement so they could be stored in the same array.

Description of Learning

I will describe this topic, in terms of what I've learned, under the headings:

1. **Technical**
2. **Personal**

1. Technical

a. Gui's:

- I've learned how to develop GUI's in java using the Swing and AWT libraries.
- I can use many different components such as buttons, labels, spinners, combo boxes, menus and scrollbars.
- I can draw graphics on the GUI. For example:
  - The fretboard for the guitar.
- I can dynamically redraw GUI's. For example:
  - The user copies and inserts bars into the GUI's tablature wherever they chose.
  - The user can open up an existing tablature and it is redrawn to the GUI.
  - The fretboard is automatically resized exactly to fit the window as the user resizes the window.
  - Bars are redrawn when their size changes due to note duration changes and changes in the time signature.
- I can use layout managers for the GUI such as Swing's BorderLayout, BoxLayout, FlowLayout, and the more complex GridLayout for the invisible buttons used to read the users note selection on the fretboard.
- I can layer GUI components using JLayeredPane. For example, I have the graphics of the fretboard on the lowest level and put the invisible buttons for the user's note selection layered on top.

b. Write code in Java:

- I had no experience in programming with Java before the project. Now I have a good grounding in it due to scale of the project, and I'm comfortable with its syntax.
- I am comfortable at writing classes in Java and have a good understanding of them. I have twenty classes in the finished product. They all have their own distinct purpose in what they do.
- I can reference objects correctly. For example, when a user selects a note on the tablature to change it, I have to know which notes object has been selected in order to display all the notes in it on the fretboard. If the notes are replaced on the tablature, I need to know which tablature object on the GUI that needs to be updated with the new notes.

c. MIDI

- I have a good grasp of generating sounds with MIDI in the Java Sound library. My project can use MIDI to play back any of a user's tablature with complete accuracy in terms of the notes used.
- I can calculate the timing of when the MIDI is played with complete accuracy. This is crucial when playing music, as it's all about timing.
- I can select the correct instruments the user wants to use from the sound bank, and play back their tablature with it. So, I can produce what sound I like with the sound banks and know how to use them.

d. Object Orientated Design

- Working with Java and developing in Eclipse has given me a better grasp on OOD. Related data are grouped in their own classes, and the methods associated with them to work on that data. Each class has its own defined role and responsibilities specific to it.
- As a result, my designs are more simplified and more robust. They are easier to debug because the classes are responsible for unique data and the unique methods that act upon them. So, less is likely to go wrong when one class is controlling the actions upon its data, and it's easier to trace where the source of a bug is as a result.

e. Eclipse IDE

- I am comfortable using the Eclipse environment for developing software. I have also used it for my third year project, computer science and python projects.

f. Serialise Objects

- I can serialise objects for storing data. In this project, this is used for saving a user's tablature. Also, I can de-serialise objects and restore the data. In this project, that means I can open a user's previously saved tablature by using the de-serialised objects. A pivotal aspect of software is to be able to save your work and be able to continue it at another time. I can do that with Java.

2. Personal

1. Time Management

I'm better at being able to schedule my time into the different things I need to do. For example, my course has a lot of practical work in many different subjects including this project. In order to fit everything in and be able achieve the best quality of work I can achieve in all of it, I need to be able to decide for how long do I concentrate on one subject, when and for how long.

Some sacrifices have to be made along the way, but to achieve an overall high level of quality in each subject this has to be done. I need to be able to say, "I have given this

amount of time to this subject, now I need to move on to the next subject or else it will suffer in quality." Even if that means sometimes, the work on the subject your leaving is not quite at the level of quality you wanted.

When you're in your final year, especially with high workload, you need to be able to spread yourself over all the subjects you need to do. You can't just decide to concentrate on a few at the expense of others, especially if you want a high grade average at the end of the year. It's a balancing act. You need to be able to structure your time to do it all, and know when to stop working on the subject your on and move to the next.

2. **Prioritising**

You need to able to way up all of the work you need to do, the time you have to do it, and be able to prioritise it. In this project, that meant not being able develop the note effects, even though they were part of my core features. It came down to a choice of making what I had done better by allowing the user the ability to save and restore their work, or trying to implement the note effects with the short time I had left. The latter of which, I may not have had enough time to implement properly, anyway.

While it would have been great to fully complete my core features, which was actually more important? Run the risk of not being able to implement note effects in the time left? That meant users can't save their work, which ruins the longevity of the product. The note effects may not have been properly done anyway. This would leave me at the point I was at to begin with, but with the time I had left wasted and nothing to show for it. Or concentrate on saving and restoring their work? This ensures the longevity of the use of the product and was almost guaranteed to be implemented, as it was the easier to implement.

It was more important for the quality of the product that a user could save the tablature they were working on, and restore it to work on it at later time. Also, that they could choose the instrument and key they used for playback in a user friendly way. I used menus for this choice. Saving, restoring data and even using menus was new to me, so they would themselves take time to code. Also, I had the time to do these things and they would add a greater depth in the user's experience than what the product had up to this point. So, I prioritised those choices.

The choices worked out, and by prioritising I had a far better guarantee of improving the product, which I did.

3. **Breaking up tasks into more manageable tasks**

I had never hard-coded a GUI before, and had never used Java. So it was daunting at first to be able to get the GUI to a point where the user could enter notes on a graphical fretboard and add them to the tablature. There was a lot to do and to know in order to be able to do it.

You need to break large tasks like this into more manageable, more realistic and attainable steps. You need to think about the steps that are needed to get to the main goal. You learn the best way is to treat each of these steps as a goal within itself, rather than the main goal being the only one. It makes sense, because you have to achieve a lot of smaller tasks in order achieve the main task as it is a culmination of all the smaller tasks.

It makes you think in a more structured way of what steps you have to do, to get to your main goal, and in what order because that is the way you are thinking. This means you design your main task better. You are more efficient at reaching it with lesser problems.

Psychologically, it's less stressful and more rewarding. It's less stressful because you are only focusing on the next step at hand. You have less to think about, and it is more manageable cognitively as you are working within your limits of what you can contemplate at any one point in time. It's more rewarding because completing each step towards your main goal, is a goal and a reward in itself.

So, I simply started with being able to layout and put components on the screen using the layout managers. Then I learned how to be able to draw graphics to the screen, such as the lines for the strings of the guitar in the tablature. Then, the graphics for the fretboard, and resizing them when the user changes the window size. After that, I learned to create the grid of invisible buttons that recognise the notes selection of the user on the fretboard. I learned how to display that selection and resize the grid of buttons along with the fretboard. Then, take that selection and translate it into fret numbers and on what strings. Finally, I learned how to display them on the tablature, and so on. All these small manageable tasks and goals eventually led to my main goal – the user being able to add notes to the tablature.

4. **Self-Motivation**

I've learned to be able to motivate myself better. This becomes more important than ever with the high workload involved in doing the last year of an honours degree. It can feel quite daunting when looked at in the bigger picture. It's better to look at it in more short term tasks and goals.

In the long term you motivate yourself by knowing it will be all worth it in the end, but that's not enough. To ensure and maintain a high level of work during the year and within this project, you set short term goals to achieve that keep you motivated throughout. As a result, you are rewarding yourself constantly in the short term and therefore, keeping yourself motivated towards the long term goal.

Review of Project

1. What went right?

- The design I had for the underlying classes and data structures worked perfectly.

  The classes were Notes, Bars, GuitarTrack and Tablature (renamed MusicTabEditor). The MusicTabEditor class stores all the tracks of the tablature in an array. The GuitarTrack class has all the bars in the track contained in it, stored in an arraylist for Bars objects. Each bar had all the notes in it, contained in an arraylist for Notes objects. Each Notes object had an array that stored all the notes played at one time in the music, stored in it.

  - So, I could always associate the bars of a track with the correct track, and the correct notes in a bar with the correct bar. This was crucial in knowing when the user wanted to change notes in a bar, or delete and insert bars into a track.
  - When it came to serialising the objects for saving the users tablature, all I had to do was write the GuitarTrack object. This is because it contained all the bars of the track, and the bars all the notes. I only had to write the GuitarTrack object. Every other object contained within it is automatically serialised as a result, if they implement serialisation.

- The finished design for my core features and the GUI work as they are supposed to.
  - The fretboard can read accurately what notes the user selects, even when the window is resized.
  - The user can add notes to the tablature anywhere they want. They can edit any notes as much as they like. They can copy, delete and insert bars anywhere on the tablature. The product maintains the correct tablature at all times for the user.
  - The user can decrease the note durations as much as they like. They can increase the note durations if it is allowed, due to what empty notes are in the bar that can be used to make up for the increased duration. This is the correct behaviour. The bars will increase and decrease in size due to these changes, and it does so correctly.
  - The user can set the time signature of any bar if it is allowed. You can't set a time signature lower, if there aren't enough empty notes in the bar that you can remove to allow for it. This is the correct behaviour.
  - The playback of the tablature plays the users tablature exactly as written by them, and from anywhere in the tablature they want to play it back from.

- The user can set the playback speed of their tablature with any tempo they choose, up to 320 beats per minute. The exact speed is always accurate.
- The user can choose any of the available instruments or music keys to playback their tablature in. What is chosen is played correctly.
- The user can save tablature and open up saved tablature to continue working on it.

2. What went wrong?

- Getting the GUI to display components properly was a struggle some of the time. If a components size wasn't declared outside of the creation of the object itself, sometimes the object displayed out of place or not at all.
    - Trial and error helped me fix this. I'm still not quite sure of the logic of it because some components display fine when their size is declared in the creation of the object.

- Getting the tablature to display correct was a problem. When more tablature objects where added to the GUI, they didn't eventually disappear out of the visible window. As a result, the vertical scrollbar wouldn't appear. Each tablature object would squash into itself when more was added in order to fit into the visible window.
    - To fix this I had to set each tablature object's maximum size when I created them. That size was set to what the preferred size of the tablature was, which I had to set separately also.

- Trying to keep the correct notes in a bar when the user increased the duration of a note caused me a lot of problems. I wasn't able to calculate correctly what notes should be in the bar afterwards. This is because you have to remove empty notes after the note being changed in order to make space for the increased duration. You include the old duration of the note being changed in this calculation. The duration of the whole bar (determined by the time signature) has to be equal to what it what it was before the change.

  The empty notes you remove may leave some duration left over in the bar to fill up, after you get enough to make up for the notes new duration. For example, the Note's old duration is 4. I need to change it to 16. The empty notes after it, in sequence, have durations of 1, 1, 1, 1 and 16. Including the notes old duration and the empty notes, this adds up to 24 with 8 left over. This is enough, but after I've removed the empty notes what empty notes do I add into the bar to make up for the leftover duration?

  I could just enter in any notes with standard durations that add up to what's left over. This would be one note with a duration of 8 in this case. This scenario has a

problem. This will potentially add in more or less empty notes, of different durations, that were not their originally in the bar. In this case, less notes and of a duration that wasn't there. It may confuse the user as to why those notes are there, and remove notes of durations they intended to be there. They may have to change these new notes which would be tedious, and it would just confuse them. It's not consistent with what notes and of what duration were in the bar to begin with.

  - To fix this, it turns out that whatever empty notes you remove, including the old duration of the changed note, perfectly add up to whatever could potentially be left over in the duration of the bar. So, the correct empty notes with the correct durations can be added back in to keep the consistency of what was in the bar beforehand. In my example, 8 was left over. So one note with a duration of 4, and four notes of a duration of 1 will be added back in to make up for the leftover duration. These are the notes that were in the bar beforehand.

- When a user was replacing notes on a particular tablature object, sometimes the notes would be changed on a different tablature object if there were many on the GUI. This was because the wrong tablature object was being referenced when replacing the notes.
  - To fix this, I had to give tablature objects each a unique id. Each notes added to a tablature object would have a reference to its id. So, when I had to replace the notes, I knew exactly which tablature object they had come from and could replace them on the exact tablature.

3. What's still left to do

- Of my core features, I only did not get to develop and implement the note effects, as it was impossible to gauge beforehand how much work the core features would entail. As a result, I did not have the time to do the note effects, as the core features were bigger in scope than could have been anticipated. In order for the product to be of higher quality, time had to be taken to implement the extra features mentioned on page 9-11.
- The discretionary and exceptional features of the functional specification I did not really get to go near. These in hindsight were unrealistic due to the high workload on the course.
  What I did get to do out of the discretionary, was part of the metronome. I can set the tempo. It is being used for the playback speed of the tablature.

4. If starting again, how would I approach it differently?

I think my design would be better and more refined. I've improved greatly in OOD and I'm far better at it now than when I started, albeit still with a lot of room for improvement.

Other than that, I think the way I approached it was generally very good. I broke my core features into smaller tasks. It gave me a more structured approach in arriving at the main goals of the core features. I managed my time well. I designed and developed the project in an OO way to the best of my ability, and that design has not caused me any real problems besides bugs here and there.

5. Advice to others attempting a similar project

- I recommend using the book *Head First Java Second Edition.*
  This book has all the basics of what you need for creating GUI's, using MIDI and saving and restoring files which I used for the tablature.
- Oracles online documentation for Java is also excellent.
- Research thoroughly what tablature, time signatures, note duration and generally how music is written and what components written music is comprised of.
- Break your tasks into sub tasks. Discover all the steps you need to reach your main tasks. This also gives you a more structured and certain path towards your main goals. Your design will be far more efficient to create and maintain.
  See each step as a goal to keep you motivated and rewarded. This will keep you motivated in the long term.
  Always keep an eye on your main goal so you know where you're headed, to avoid ending up with a product that doesn't quite match the requirements.
- Only learn what you need to know in relation to your project. Don't get side tracked learning things that are not relevant or specific to what you need to know. It only wastes time.
- Use the internet to supplement any documentation you have about what technologies you use, or any information you need to know about your projects subject content. The internet is an invaluable resource, and you may find the answer you're looking for far quicker than trying to find it in books.
  The information is nearly always there at a click away. Use it.
- Manage your time properly. Try to set realistic targets of what you want to achieve with a particular part of your time. It motivates you to work harder and concentrate more.
  Don't become disheartened if you fail to meet your targets in the time set, and don't try to squeeze in what you haven't done with what you do next time. Otherwise, you may put undue stress to on yourself which is counterproductive. Missing targets happen, as you can never truly know how long something will

take you to do if you have not done it before. Re-adjust your targets next time to compensate for what you didn't get done the last time.

6. Outcome of Technology choices

The technologies I chose worked out perfectly for me. I could do relatively comfortably what I wanted to achieve with Java, Java Sound, Swing and AWT libraries.

Getting Swing to display some components felt a bit awkward at times, as some components didn't display properly depending on what way their size was set and where it was set. It seems a little inconsistent and unintuitive in that way.

I used Eclipse as my IDE. I can't recommend this enough. It's an incredible tool that makes your life so much easier. I hadn't coded in Java before and it helped me learn incredibly quickly and how a Java programmer codes. Every object you create, Eclipse will show what methods can be applied to it when you try to use a method. It gets you used to the style of coding in Java, as you can see how data is associated with its own specific classes and how the data is retrieved. It also makes development so much quicker and more of a joy to do. It takes out a lot of the hard work, and lets you just concentrate on what you have to do.

I definitely made the right choices in the technologies I chose. Java is very logical and intuitive to code with, so extracting the functionality I needed was a relatively smooth process.

Acknowledgements

I would like to thank my supervisor and lecturer, Paul Barry, for all his patience and guidance throughout this project. For giving me feedback on the project sometimes outside of college hours. I wouldn't have been able to pull off this project to the standard it is without his help.

Also, thanks to the other supervisors Christophe Meudec, Joseph Kehoe and Nigel Whyte who gave feedback on our project presentations and offered any help if needed.