Manchester Metropolitan University

BSc (Hons) Computer Studies

Axe Class

A Computer Game for Learning the Guitar

by John Rees

Dissertation for Honours in Computer Studies

Supervisor: Dr Andy Nesbit

Acknowledgements

I would like to thank Dr Andy Nisbet for his patience and enthusiasm with this project, particularly in the early stages. I would also like to thank Dr Lida Nejad and my family for spurring me on during the tough times.

Declaration

No part of this project has been submitted in support of an application for any other degree or qualification at this or any other institute of learning. Apart from those parts of the project containing citations to the work of others, this project is my own unaided work.

SIGNED _____ DATED ____/____/_____

Abstract

This report endeavoured to research methods used in computer-based guitar pedagogy. Then consequently provide a detailed commentary on the process of developing new methods and techniques that could be applied to a guitar learning game. After sufficient planning, the primary goal of the research effort was to produce a Computer Assisted Learning Game for the guitar. This was achieved by building iterative prototypes using ActionScript, Flash and Java technologies. Once these had been created, a desktop and a mobile game called 'Axe Class' were formed, to culminate the ideas that had been developed in the prototypes.

Table of Contents

List of Figures

List of Tables

Terminology

AS3	*Actionscript 3, a programming language developed by Adobe and based on ECMAScript*
AIR	*Application Installer Runtime, a cross-platform environment for building desktop applications using AS3 or HTML*
Array	*An indexed data structure of data values*
Bend	*A vibrato produced on stringed instruments by pushing the strings*
Chord	*A collection of notes played together on the guitar neck.*
Fitt's Law	*An equation which predicts the time required for a human to point to a target area*
Frequency	*The number of occurances within a given period. Sound frequency is measured in Hertz (Hz), the number of cycles per second*
Hammer-on	*A technique that is achieved by playing two or more notes, by sharply striking a fret position with only one stroke of the string*
Harmonic	*A musical note played by placing a finger over a vibrating string at a certain point on the guitar neck*
MIDI	*Musical Instrument Digital Interface, a protocol that permits communication via instruments and software by transmitting commands instead of audio data*
MySQL	*A popular open source SQL (Structured Query Language) relational database management system. Commonly used on the internet and well implemented by PHP.*
PHP	*Pre-Hypertext Processor, an open source server-side scripting language.*
Pick-Up	*A magnetic coil that sits underneath the strings on the face of an electric guitar, and converts the strings' movement into an electric signal. A divided pickup is used on certain MIDI configurations and seperates the MIDI signal into separate channels.*
RAM	*Random Access Memory, an integrated circuit that can temporarily hold data*
RMTP	*Real Time Messaging Protocol, a proprietary protocol developed by Macromedia used to stream data over a network*
Synaesthesia	*An emotion experienced when different sensations are coupled, e.g. sound and colours.*
SysEx	*A System Exclusive message is a MIDI signal that is specific to the equipment that is using it. Yamaha's messages contain nine parts and the data is in hexadecimal format.*
USB	*Universal Serial Bus, a standard port used for connectiong peripherals to a computer.*
XML	*Extensible Markup Language, a general purpose markup used for describing many different kinds of data*

Chapter 1. Introduction

1.1 Background for the Project

Being both an avid gamer and diffident guitar student, I was very enthusiastic to be involved with this subject matter. After failing to learn to play using several instructional books, mainly due to a lack of sense of development and ultimately motivation, this topic was of great personal interest. I had always surmised that there must be a way in which computers could help the learning process, but my experience with teaching software up until now has been predominately lacklustre. Crucially this game is something that I would want to play myself and this will spur me on to produce the best piece of software I can, through the eyes of both the developer and the user.

Gershenfeld (1999) identifies that people started playing music because it was the only way to hear it, then when mass media was introduced society was split into two groups, those who are paid to be artistically creative and the much larger group who passively consume their output. He concludes that reducing the effort to learn to play an instrument will enable far more people to creatively express themselves and that "improving the technology for making music can help engage instead of insulate people".

In recent years, hundreds of developers producing computer-based courses have attempted to replace printed books. E-learning has been the focus of an enormous amount of research, yet there have been few attempts to develop a complex computer interaction through the physical use of a musical instrument for the sake of learning (Liarokapis, 2005; Galarneau, 2005; Gunter et al, 2006). Xin et al. (2007) noted that the benefits of applying physical interaction to electronic games are seldom documented.

1.2 Objectives

The aim of this project is to demonstrate methods of teaching guitar with a computer game. Several prototypes will be produced, including a mobile game. A game called Axe Class will then be created using the knowledge acquired from the previous stages.

1.3 Aims

The goals that should be achieved during this project are listed below:

- Research tools, literature and software available
- Produce a virtual fretboard to help practice learning note positions by sight and tone
- Implement a moving line stave which will help emulate reading tablature and/or sheet music
- Provide data to the application externally i.e. XML to allow for expansion

1.4 Boundaries

- The software will not be designed to teach complicated songs or techniques, more a tool to enrich learning and emphasise motivational development
- The player doesn't complete the game, but they may advance levels
- The game is not intended to replace human tuition but rather to complement it

1.5 Deliverables

The deliverables for this project are several iterative software experiments and a computer game, which will be provided on a disk attached to the project report's cover.

1.6 Project Schedule

A Gantt chart for this project is included as Appendix B with this report.

1.7 Overview

The structure of this report is a reflection on the order in which the areas were covered. It is divided into the five chapters listed below:

- Literature Review- presents a review of research conducted into pertinent areas related to this project.
- Design– states the identified requirements, explains which development methodology shall be used and discusses the control system alongside the user interface
- Implementation and Testing- describes the work undertaken whilst developing prototypes and details of the testing procedure.
- Evaluation- a critical analysis of the project and a discussion related to what extent it was a success.
- Conclusion- summarises the report findings and acknowledgement of limitations, finished with suggestions for future research.

There is also an appendices section attached with this report that contains the following items:

- *(Appendix A)* Terms of Reference
- *(Appendix B)* Project Schedule/Gantt Chart
- *(Appendix C)* User Manual for Axe Class
- *(Appendix D)* Extract from Yamaha EZ-AG Manual
- *(Appendix E)* Source code for the project's software

1.8 Relevance to Degree

This project will provide the opportunity to apply the knowledge I have acquired throughout my Computer Studies degree; with particular focus on Advanced Programming and Human Computer Interface Studies.

1.9 Chapter Conclusion

Now that the aims and objectives have been declared, in accordance with the project overview (Section 1.7) the next phase of the report is the review of literature.

Chapter 2. Literature Review

Before planning the design of the software to accompany this report it is necessary to research the themes that might be relevant. This chapter begins with a generalised overview of the subject and then proceeds to scrutinise specific pertinent topics more thoroughly. Similar software and literature will be particularly useful as an insight into how the prototypes should be developed.

2.1 Background

Prior to expansive research in the subject, it was once believed that teaching is a necessary condition for learning (Ruben, 1995) and that it should be segregated from playing (Bates, 2000). Since this hypothesis was formulated, many developers have led themselves to believe that because the educational content is housed inside a game, players will be motivated and excited to learn (Gunter, Kenny, Vick 2007). Bates (2000) concludes that teaching material is in great need of a more creative method of delivery.

2.2 Related Material

There is a ubiquitous selection of education software available, yet the majority of these titles offer a simple linear one-way teaching style, the same level of interactivity as a book. To be classed as a game, the educational software needs to offer interactive metaphors of knowledge (Denis, Jouvelot 2004). Well-contrived music software, even if it is not designed to teach, will help to stimulate a positive attitude towards music and encourage self-expression (Bates, 2000).

Interactive music software is now evolving from its infancy, and currently several exciting experiments are being developed, for example, the I-Maestro project (2006). I-Maestro is an all-in-one learning tool that will facilitate new technologies to help broaden learning through a co-operative network environment.

2.2.1 Music Games

For the last twenty years there has been a commercial market for games that have been produced with the aim of teaching the user to learn about music, or at least foster their interest in the subject (Percival, Wang, Tzanetakis 2007). It was not until more recently however, that the music genre was widely recognised as a category of game. The genre is unique in that generally it does not conform to any sets of rules in the way

others have, no other genre has experimented with gameplay ideas as prolifically as the music genre (Pichlmair and Kayali, 2007). Pichlmair and Kayali determined that a music game could be categorised as either Rhythm Based or Instrumental with some exceptions such as the musical puzzle minigames in the Zelda Series (Nintendo).

In the mid-1990s developers started to realise that there was mass-market appeal for rhythm-based games, and 1996 introduced several big hits such as PaRappa the Rappa (NaNaOn-Sha) (Figure 2.1) and Beatmania (Konami). Whilst these weren't designed as learning games, they do elude to a music-learning environment. PaRappa implements a 'Simon Says' style teaching method, where instructions are issued by a teacher during a song and the player has to press the appropriate button to perform the action in time with the music.

Figure 2.1 - Parappa the Rappa Screenshot

Both of these games encourage the user to interact by asking them to identify and follow the beat of the songs played, as the player becomes more proficient at the task the level of difficulty increases. Several years later some more advanced titles with dedicated peripherals were introduced including Donkey Konga (Nintendo, 2003) and Guitar Hero (Harmonix Music Systems, 2005) series. Other notable titles include Rez, a game based on the concept of synaesthesia (when different sensations are coupled, in this case, sound and colours) and Vib Ribbon, a music platformer hybrid game that was stored in the console's RAM, which calculated the shape of the level from a song on the player's music CD.

Harmonix's Rock Band, is played using included peripheral instruments. The game allows players to form a band with a drummer, guitarist and singer. In the same vein as the Guitar Hero series, Rock Band does not proclaim to teach the user to play guitar, as the controller is designed enhance the sense of immersion and is

not marketed as an instrument; nevertheless the singing and drumming sections of the game may be very close representations of reality. This is consistent with Harmonix's credo; that most music fans may not have the time, patience or talent to learn a musical instrument yet there is still the opportunity for them to get the same joys out of playing music. Despite this, they do argue that the Guitar Hero series helps the player's sensitivity to rhythm and independent hand usage (Manjoo, 2007).

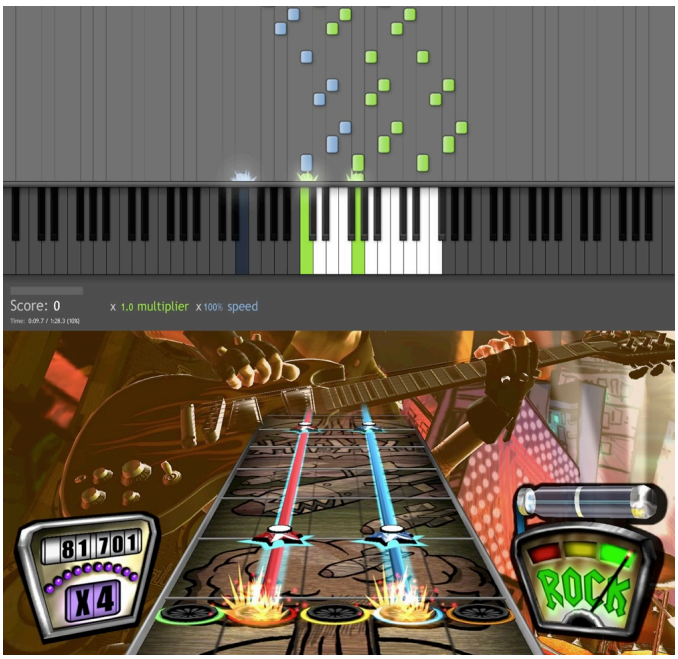


Figure 2.2 A Comparison of Synthesia (Top) and Guitar Hero 2 (Bottom)

2.2.2 Educational Music Games

As a result of their research on motivational factors in music games, Denis and Jouvelot (2006) produced the Java based game Pads 'n Swing (Figure 2.3). The game breaks normal boundaries normally imposed by music games and actively encourages users to improvise with the jazz backing tracks. The scoring system is based on the user keeping time with the rhythm of the game and it is controlled using gamepads. Whilst the user interface is appealing, after obtaining feedback from users who tested the game it was concluded that its design not intuitive enough. There was no language barrier as the interface is almost exclusively graphically based; however, without any initial tutorial or instruction manual it was very confusing to play.

Figure 2.3 - Pads 'n Swing Screenshot

Synthesia (Originally named 'Piano Hero') was inspired by the Guitar Hero series (Figure 2.2) and was designed for players wanting to learn songs to play on the keyboard. The concept of the game is to juxtapose a graphical representation of the player's keyboard with falling bars that land onto the specific keys and the player then presses the corresponding notes in time with the music. This interface emulates the Piano Roll, a popular 20th Century method of reproducing and playing music on the piano with a roll of paper turning to a specific tempo that had perforations to represent the note and its duration. Figure 2.4 helps to illustrate the similarity between the piano roll with this software.

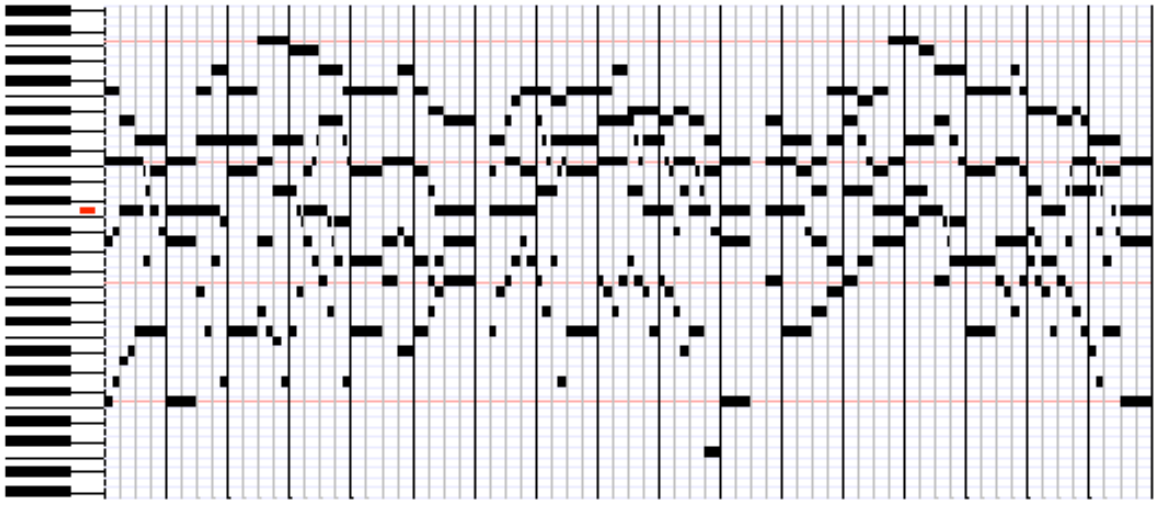


Figure 2.4 - Piano Roll Diagram

The major features of this game are that the user is learning with a real instrument, any MIDI file can be imported and played and that the game keeps track of the player's 'score' throughout the game giving an evaluation of the performance the moment a song is finished. The keyboard is a simple instrument to make a learning game for because of the binary nature of the instrument; a note is either pressed or not pressed. With a guitar there are a many more intricacies but elements of this game could definitely be applied to this project.

The Internet offers many games for learning the guitar, but few of them are designed to be playable rather than behaving like an interactive book. GuitarGames.net is a good example; the site provides several Mini Games for learning about music theory. Despite the gameplay repetition that is present in some cases, they are fun to play and an effective tool for tracking the player's progress.

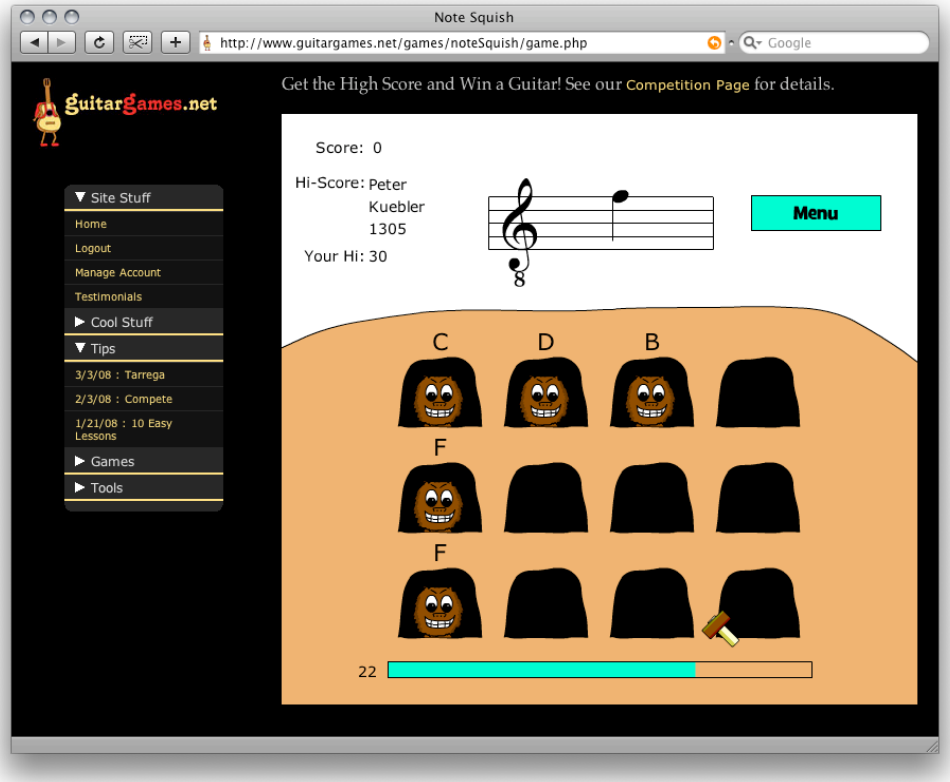


Figure 2.5 - GuitarGames.net Note Squish Game

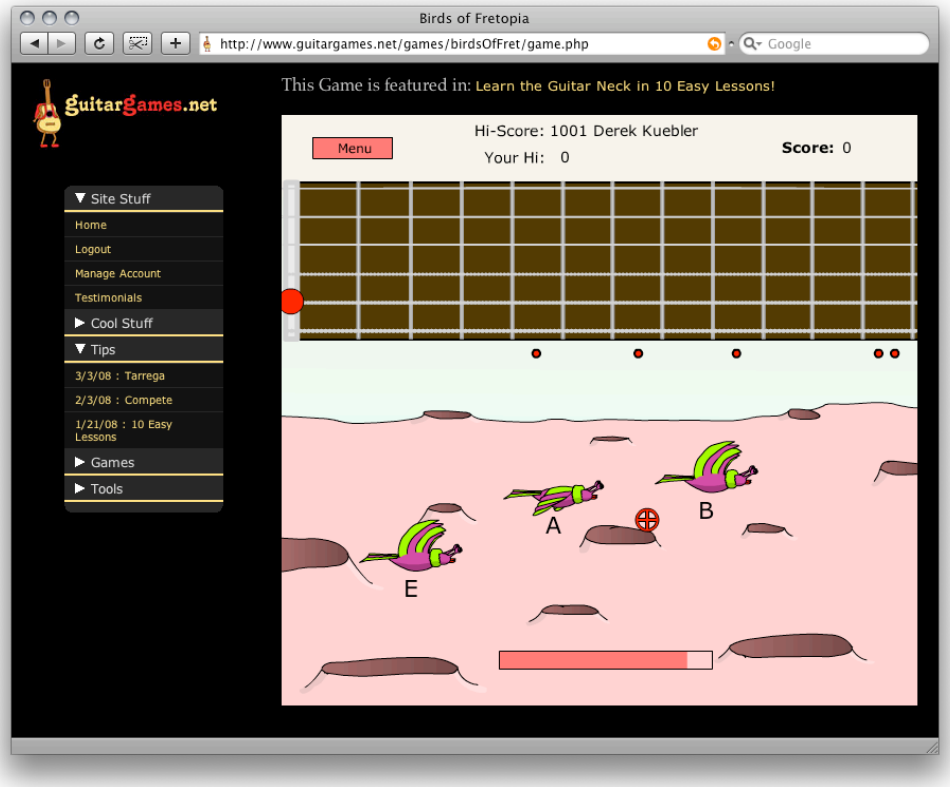


Figure 2.6 - GuitarGames.net Birds of Fretopia Game

2.2.3 Applications

Guitar Pro is one of the 'all-in-one' solutions for learning guitar, it will play a MIDI song whilst scrolling through both TAB and Standard notation simultaneously. There is a virtual keyboard and guitar fretboard to show the correct positions during the song, this is very effective but it doesn't show finger positions like the Learn 2 Play (2001) series does for example.

Absolute Fretboard Trainer has more specific teaching objectives; its purpose is to teach the user every note on the fretboard. Again, it offers no interaction with a guitar or MIDI device, in spite of this it still has an effective feedback system. Whilst the user is being tested on the various learning exercises, it creates a colour-coded map of the user's proficiency and knowledge for each string and fret. It is calculated from the time taken to respond to questions about the fret position and whether or not it is answered correctly.

The system could be improved if it were to automatically test users on areas they score badly on. Currently the documentation suggests that the user should manually move the fretboard area (a draggable box) over their weaker points to get tested on them in future sessions.

2.3 Game Design

Game design guru Sid Meier defined games as a "series of interesting choices".

It is commonly agreed (Bates, 2000; Pichlmair M, Kayali F, 2007; Dannerberg et al, 1989) that music tuition software is much more effective when combined with a human teacher's support. There are several arguments for both CAL (Computer Assisted Learning) and traditional human methods. While computers can offer endless patience, a self-paced environment and 24 hour access, it is still very difficult to replace all of the intricacies of a shrewd human tutor e.g. the ability to ask spontaneous questions (Bates, 2000) and there is the danger that the player may pick up bad habits early on in their development which will be hard for them to forget later on**.** If the game isn't well structured or very intuitive, again there may be a requirement for a teacher/facilitator in order to maintain the players interest by providing exploration paths and avoiding random expression (Denis, Jouvelot, 2004).

When designing any game, the best way to build it is by repeatedly directly experiencing the game as it is made (Salen, Zimmerman, 2003); this is the process of Iterative Design. It puts emphasis on prototyping, evaluating and refinement. The game should have a simple prototype as early as possible, not to show how the game will look but rather how it will interact. The Game Object Model (Amory, 2007), dictates that a game needs to be relevant, explorative, emotive, engaging and include complex challenges. It also states that there should be no black and white win/lose situation but more complex 'non-confrontational' outcomes.

2.3.1 Cognitive Design

Some researchers feel that an educational game's ineffectiveness can stem from the creator not having clear intentions of what it will teach (Gunter, Kenny, Vick, 2007). By achieving the wrong balance between learning and play, educational game-based software has gained many negative connotations (Becta, 2006). Denis and Jouvelot (2004) contend that gameplay is often the afterthought in development of educational software which makes it very boring to play and difficult for the player to stay motivated. Many consider gameplay to be the most important indicator of the quality of a game. According to Draper (2000), fun is not a property of software but instead a relationship between the software and the user's current goals.

There is a danger of a learning game becoming a simulation, which on its own is simply an evaluative tool and doesn't guarantee that the learner will acquire conceptual knowledge. Real understanding may only be achieved when the simulation is used together with other activities that support cognitive development (Kieras, 1997).

Players have not only come to expect a certain quality and playability of a game but their actual learning traits has come to 'evolve'. Prensky (2001) highlights the following characteristics in today's gamer-learners:

- A preference for graphics over text
- A random, informal approach to information
- A need to stay connected to peers
- A desire for an immediate payoff
- An understanding of information being a commodity

In order for the player to be immersed in the game, interaction and engagement are required (Laurel, 1993). The following characteristics are essential to the design of engaging environments (Jones, 1998):

- Task that we can complete
- Ability to concentrate on task
- Task has clear goals
- Task provides immediate feedback
- Deep but effortless involvement
- Exercising a sense of control over our actions
- Concern for self disappears during flow, but sense is stronger after flow activity
- Sense of duration of time is altered

2.3.2 Flow Theory

Csiksczentmihalyi (1988) describes flow as "being completely involved in an activity for its own sake. The ego falls away. Time flies. Every action, movement, and thought follows inevitably from the previous one, like playing jazz. Your whole being is involved, and you're using your skills to the utmost."

Games foster play, which produces a state of flow, which increases motivation, which supports the learning process. (Paras, Bizzocchi 2006). However, Csiksczentmihalyi states that if the difficulty level is too high then the player will become overly anxious and not experience flow, a similar situation applies if the challenge is too easy except the player will become bored. The theory is a suitable guideline for game development, in order to adhere to the flow theory; a learning environment must closely match each student's skill level, and provide tasks with clear goals and immediate individual feedback (Paras, Bizzocchi 2006).

Bradshaw (2005) defines a successful game as a series of goal related skills increasing in difficulty until attainment of the overall goal is achieved. Therefore, in order for the theory to be effective, the activity should be structured so that the user can increase or decrease the level of difficulty being faced, so that his skills can be matched with the requirements (Csiksczentmihalyi, 1988). Guitar Hero is a prime example of a game utilising the principles of flow, as the player improves they will unlock harder levels and until the player returns to the early, introductory levels it is not obvious to them how much they have improved at playing the game. This is, in turn, demonstrates one of the drawbacks of the flow state; reflection does not take place during the flow experience (Paras, Bizzocchi 2006).

2.3.3 Motivation and Feedback

"The single most useful factor in any practical multimedia system for students is motivation. If students could be motivated to play their assigned musical instrument exercises every day, that would far outweigh the benefit of the fanciest multimedia feedback system." (Percival, Wang, Tzanetakis, 2007).

Providing motivation is one of the most important aspects of this project, as music teachers are aware, it is critical that players are motivated to practice, especially in the early stages when they are more susceptible to getting bored or frustrated and quitting. To help to keep the player motivated they need to be actively learning, rather than fed examples. In simulation environments, the motivational pull of reality games is due to wanting to learn how to do the task, for example, learning to fly (Gunte, Kenny, Vick, 2008).

Conveniently, games are the ultimate device for creating and maintaining motivation (Denis, Jouvelot 2004). This theory is justified on a daily basis, when millions of people around the world pay to perform menial tasks in online environments in games such as World of Warcraft (11.5 million monthly subscribers, Blizzard, 2008). A guitar learning game should also be inherently motivational because the player will be motivated to learn to play the instrument, however this isn't always enough as a steep learning curve will often dissuade learners.

With self-paced activities, players tend to lose motivation, this can be assisted through working with others - there needs to be a sense of community (Isaacson, 2001). The game audience needs to be kept in mind when it is being produced, as assorted groups of people will respond differently to various types of motivation. For example, young players would be appreciative to receive coloured stars for their efforts whereas older players would prefer to be able to compare scores and compete with their peers (Denis, Jouvelot, 2004), Xbox Live is built on these tried-and-tested principles and has proven to be a very successful system. Some players might even be annoyed with the motivational aspects of a game and they should be catered for with a

separate section, but there will be a sample of players who are very good at learning but do not have the willpower to practice everyday (Percival, Wang, Tzanetakis 2007).

As systems have become more powerful, developers have come to realise that initial motivation to play and keep playing a game can be greatly affected by the quality of the sound and the graphics in the game. The player needs to be frequently rewarded for their efforts, especially early on where they there needs to be an abundance of reward for very little effort This is called the feedback loop and is what keeps the players engaged, now these are being more widely understood a large part of the designers role is to control the type, frequency and scale of the rewards (Becta, 2006).

The challenges that are offered must have intrinsically motivating qualities. Intrinsic motivation has been proved to be more effective than extrinsic; the learner must feel that they are choosing to learn. Due to the nature of a game, especially one where equipment needs to be bought, it is more likely that this will be the case. Clark (2006) contends that too many game controls e.g. countdown timers, can inhibit the intrinsic motivation. He further concludes that there are seven main factors that contribute to motivation, which are:

- Intrinsic - visualisation and problem solving appear to be closely related to intrinsic motivation and learning (Rieber, 1995).
- Autonomy - people need to feel the origin of their actions
- Self-confidence
- Challenges
- Feedback
- Goals
- Social

As described earlier, players rarely reflect on the learning that is taking place (Paras, Bizzocchi 2005), so in order to remind them there should be constant, relevant feedback on their performance. It is not the goals in the game which will motivate the player, but the individuals evaluation of their own performance in relation to those goals. Learning from the strengths of existing successful methods, a comprehensive feedback system can be created.

Intrinsically motivated learners will attempt optimal challenges but need frequent feedback (Clark, 2007). It is feedback throughout the learning process that matters, not end-point testing. Games have the added advantage that achievement is a form of assessment in itself i.e. completion in itself is proof of competence (Clark, 2007).

2.4 Guitar Research

Before building the games, it is important to have a comprehensive understanding of what it is that is being taught; therefore researching the guitar itself would be prudent. This section will provide a summary of the findings from this research.

In western music there are two types of guitar, the hollow-body acoustic and the solid-body electric, there are also hybrids such as the electro-acoustic but for the sake of simplicity these will be ignored in this chapter. The acoustic guitar relies on the vibrations of its strings being amplified by the hollow chamber inside its body whereas the electric is very quiet unless it is plugged into an amplifier, so that the signal generated from its pickups can be processed and the sound can be played from a speaker. It is most likely that an electric guitar will be used instead of an acoustic in this project; the next section of this chapter will explain the specification of the electric guitar.

2.4.1 The Electric Guitar

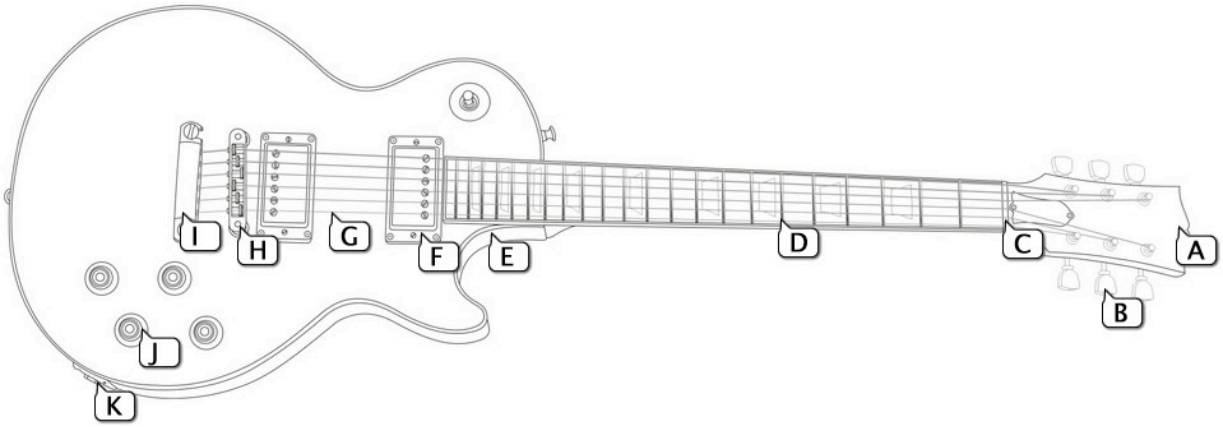


Figure 2.7 - The Structure of an Electric Guitar

The basic structure of an electric guitar is shown above. The standard configuration is 6 strings [G], with a different diameter, ranging from widest at the top to narrowest at the bottom. The strings' tension is changed with geared pegs [B] at the headstock [A] on the guitar. Varying the tension in the strings allows them to sound a different pitch when struck. The most common tuning is called Standard Tuning and it is represented in Table 2.1.

Table 2.1 - Standard Tuning

String Number	*Note Value*	*Frequency (Hz)*
6	***E***	***84.407***
5	***A***	***110.00***
4	***D***	***146.83***
3	***G***	***196.00***
2	***B***	***246.94***
1	***e***	***329.63***

[C] is the nut and [H] is the bridge, their job is to suspend the strings above the neck [D]. Metal dividers called frets divide the neck into regular (non-linear), to sound a note the finger should be pressed in-between two frets whilst the corresponding string is struck. The neck connects to the body of the guitar [E] where the strings pass over magnetic pickups [F]. Pickups come in different shapes and sizes, they act as transducers whereby they react to the motion of the guitar strings and induce an alternating current that can then be amplified. [J] are the control knobs, these can alter the way that the pickups behave and the volume of the guitar when it is connected to an amplifier via the input port [K].

2.4.2 Formation of Notes

The different sounds that a guitar makes when its strings are struck can be measured in Hz, this measurement is called the pitch of sound. The pitch of a vibrating string depends on

- The mass of the string, the thicker strings vibrate slower creating a lower sound or lower frequency.
- The tension of the string, tightly stretched strings will create a higher sound or higher frequency.
- The length of the string, the shorter the string the higher the pitch.

2.4.2.1 Notes

To identify these different pitches and assist repeating a performance of a piece of music, a scale of twelve pitches (notes) was devised, called the Chromatic Scale (Table 2.2). The twelve notes compose an octave, notes from a higher octave are doubled.

Table 2.2 - The Chromatic Scale

C	C#	D	D#	E	F	F#	G	G#	A	A#	B
	Db		Eb			Gb		Ab		Bb	

2.4.2.2 Chords

When certain notes are played together they form a chord. Chords are described in the same letter notation that is used to identify individual notes, i.e. A, B, C, D, E, F and G. Each chord can have many derivatives, for example the A chord has eight basic variations A+, A7, A-, A-7, A6, A-6, A+7 and A9 (Liarokapis, 2005).

2.4.3 The Fretboard

The guitar fretboard can be described by two variables - the strings and the frets (Dambrauskas, 2003). When the strings are pressed into the frets they will produce different sound frequencies. This is because of the change in tension and length of the string from the guitar body to the finger. If the guitar has been correctly tuned then the frequency of the sound should match one of the notes in the chromatic scale. Any given note produced by the guitar may have as many as six different positions on the fretboard. Because of this characteristic, developers have found it extremely difficult to generate playable music in their software (Dambrauskas, 2003).

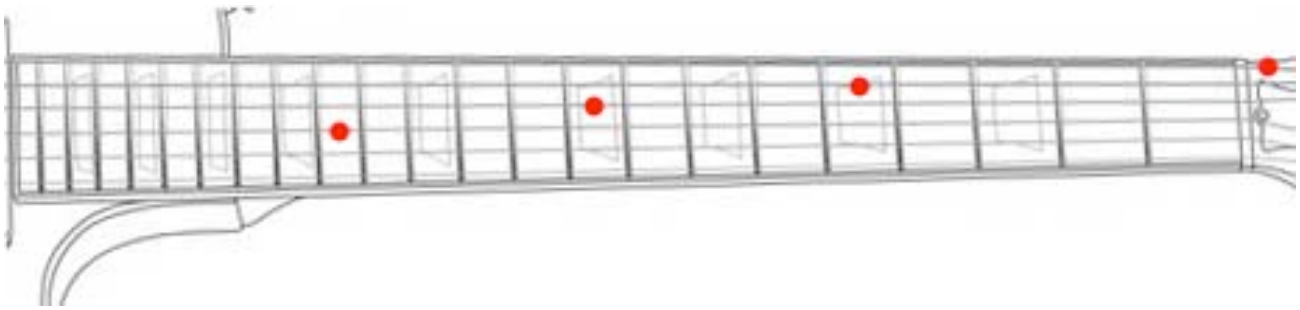


Figure 2.8 Four different fretboard positions for the 3rd octave 'E'

2.5 Musical Notation

In order to learn to play the guitar the student must have a learning device, it is possible to learn by watching and copying someone, or more experienced players can listen to a piece and then play their interpretation of it. However, if the learner is unable to do this they can work with a notation, this is a method of recording performance detail by writing it down. Various notation formats will now be examined for their suitability in the game.

2.5.1 Standard 'Sheet' Notation

The aim of a musical notation is to equipoise completeness and simplicity for the reader. A linear relationship can be observed between the complexity of the notation and the accuracy of the performance that can be interpreted from it (Cabral et al, 2001).

When execution precision is essential, rich notations are unavoidable, several performance details must be provided. This includes classical music, which is based on a score. Contrarily, the richer the notation, the less readable it is. Beginner musicians prefer intuitive notations (Cabral et al, 2001) as they can learn pieces faster and easier than they would with rich notations, generally speaking precise detail does not concern them at this stage. Eminent guitarists Jimi Hendrix (History Link, 2005) and Tommy Iommi (Dümpelmann, 2003) famously could not read traditional sheet music notation. Further evidence of this lies with the enormous popularity the Guitar Hero game series, where the developers refuse to impose an educational style and instead want their games to act as an inspirational tool, a stepping-stone before buying a guitar and learning to play it. With feedback from tutors they have learnt that many players have been motivated to learn the guitar after playing the game, whereas in the past they would be put off at the very early stages because their initial experience was frustratingly difficult (Dobra, 2009).

The main deterrent facing a beginner guitar student is that if they want to read sheet music, they must first be aware of the various positions of notes on the fretboard. This can be cumbersome, because as explained in Section 2.4.3, the notes are repeated in different positions on the board. In the early stages of learning, beginners are much more likely to be motivated when they can play simplified versions of songs they like as soon as possible. Tablature offers this gentler learning curve, at the expense of completeness offered by richer notations.

2.5.2 Tablature

Tab is a method of mapping a sequence of notes to a set of guitar fretboard positions, it describes to the performer exactly how a piece of music is to be played by graphically representing the six guitar strings and labelling them with the corresponding frets for each note, in order (Dambrauskas, 2003). A 0 would indicate that the string is to be played open i.e. without being pressed on the fret.

2.5.2.1 Published Tab

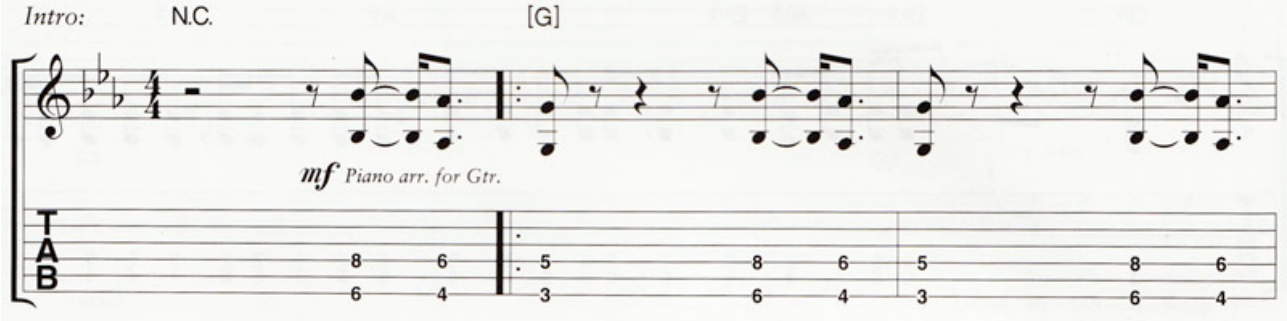


Figure 2.9 - Extract from 'Little Acorns', performed by The White Stripes, written by Jack White

Published Tablature is usually accepted as the tidiest and most readable form of tablature. It is provided with a key, although most versions are almost identical. Notation such as bends are very clear because of big sweeping lines and the lack of timing information is compensated for by printing standard sheet notation above it then lining up the notes with the tab digits. It is also common to see other features such as stem lines across the tab lines to represent time bars, and chord names printed above the tab. These additions make published tablature more of a hybrid sheet tab notation than the raw text tab that is easily found on the Internet.

2.5.2.2 ASCII-Tab

ASCII Tablature is the most popular form of notation freely available to guitarists today, mainly because of the Internet and fixed width fonts, allowing it to be transcribed and distributed very quickly. This has resulted in a plethora of people's interpretations of songs. In recent years several large music publishers have ordered the most popular sites to cease and desist because they believed that the sites were profiting from distributing copyrighted material. The sites' users argued that the tablatures did not contain rhythmic information of the song and were just inaccurate interpretations of the songs.

Unfortunately, because tablature is inherently human readable and not machine readable it is not strictly defined. This lack of standardisation has made it difficult for software to accurately interpret raw tab files. The main criteria for tab are six lines to represent the strings and digits (typically spaced with hyphens) to represent the frets. Other characters are also often used, such as `b` to represent a string-bend, or `h` for a hammer on.

Efforts have been made to derive this notation into a more computer-friendly format whilst ensuring that it is still human-readable, these shall now be examined.

2.5.2.3 Tablature Derivatives

In an effort to combine the simplicity of tablature with the comprehensiveness of standard notation several notations have been produced in the recent years, notably the era of home computers. Their main aim is to present a notation that is both human and computer readable.

One of the most popular projects was ABC notation. The project was abandoned in 2000 and in comparison to ASCII-Tab the uptake was relatively scarce. A sample ABC file is shown below -

```
T:The Legacy Jig
M:6/8
L:1/8
R:jig
K:G
GFG BAB | gfg gab | GFG BAB | d2A AFD |
GFG BAB | gfg gab | age edB |1 dBA AFD :|2 dBA ABd |:
efe edB | dBA ABd | efe edB | gdB ABd |
efe edB | d2d def | gfe edB |1 dBA ABd :|2 dBA AFD |]
```

The lines proceeding with a character followed by a colon are metadata -

- T: is the title
- R: is the genre
- M: is the default time signature
- L: the default note length
- K: key

Similarly to Standard Notation, the prerequisite of this system is that the reader must already know how to play the chords on their instrument. It is not as elegant as tab and is not quite as approachable. It is also unable to describe single notes and complex timing.

2.6 Conclusion

Table 2.3 - Musical Notation Comparison

	Standard Notation	**Tab**	**ABC**
Learning Curve	Difficult and intimidating for absolute beginners. User must first learn fret positions to play the notes.	Very intuitive, almost instantaneous to learn. The player does not need prior knowledge of the positions of notes on the fretboard. Would help to make game controls easier to learn and retain, resulting in wider accessibility (Xin, 2007).	Simple to understand, user required to know fret positions.
Computer Friendly	No.	ASCII-Tab is very popular, although lacks strict standards. May be suitable for XML.	Very, designed for computers.
Distribution	Difficult to source for free. Very popular for classical music, found in most guitar books.	Readily available for free on the internet and in most guitar books. Most popular choice with modern music.	Quite scarce, although reasonably popular for folk music.
Accuracy e.g. Timing Information	Very accurate. Required for exact replication of a performance.	Reasonably accurate, although it cannot be used to determine the given duration of any note or chord.	Uses bars in a similar way to published tab. Cannot show single note information.
Instrument Specific	*No*.	*Yes*.	No.
Extra		*Suitable for use with multiple tunings*	

As shown in Table 2.3 the benefits of using tablature outweigh using other notation formats. Although in printed format it does not convey as much performance detail as Standard Notation, this can be overcome

with the use of multimedia. Multimedia can improve notation richness without the cost of losing simplicity, an example of this is Instrumental Performance Systems; these show the song performance on a virtual instrument on-screen (Roads, 1996).

Chapter 3. Design

In this chapter the project aims stated in Section 1.3 will be expanded upon in relation to the software component of this project. Following this, several development methodologies will be examined and the choices of language, platform and control interface will then be justified. Finally, the user interface will be planned and discussed.

3.1 The Premise

The deliverables of this project are more focussed towards developing and testing features and helping research in contrast to producing a single product. The software will be separated into three parts, Experiments, a Mobile Game and the Axe Class (the Main Game).

3.1.1 Mini Games and Experiments

The aim of building the mini games is to help to segment different characteristics of the game. This will broaden the project's scope and lead to focussing its research on the more interesting and preferably less documented features.

3.1.2 Mobile Game

The purpose of the mobile game is to provide a tool that can help to improve the user's awareness of the fretboard while a guitar is inaccessible for the student.

3.1.3 Axe Class/Main Game

The main game will be a culmination of all the pertinent knowledge acquired in this research effort. It will encompass notions and ideas developed in the Experiments and will hopefully assist in expanding research in this field.

3.2 Software Development Methodologies

A development methodology is the mixture of a software development model with a set of procedures and documentation that assists developers creating a system (Reed, 1995). In this section three separate development model candidates will be reviewed and consequently an appropriate methodology will be chosen for this project.

3.2.1 Criteria

The methodology that is designated for this project will need to support a procedural approach so that deadlines for project milestones can be met, whilst allowing some flexibility throughout the development phase to accommodate any unforeseen problems or delays during production.

3.2.2 Candidate Methodologies

3.2.2.1 Waterfall Model

The Waterfall Model is the oldest methodology that will be examined in this section; it was originally cited in an article published in 1970 (Royce). It outlines a very regimented approach to a project. The metaphor is based on the premise that each section of the project is completed systematically and will therefore flow sequentially into the next. In the original model the phases are undertaken in the following order (Figure 3.1).

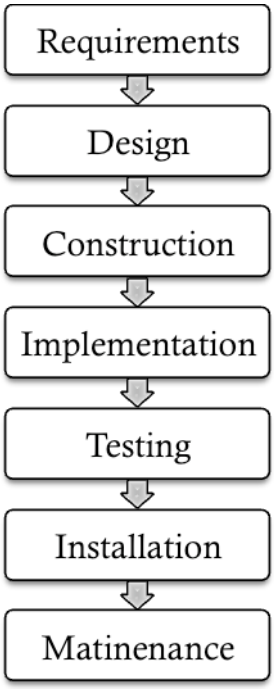


Figure 3.1 Procedure of Events for Waterfall Model

The benefits of this methodology are that discipline is enforced from the offset, emphasis is placed on documentation, time wastage is minimised at the development stage and it is easier to predict the schedule for the project. Critics of the model argue that it is unrealistic to dismiss the fact that the project will not have any unforeseen problems and that estimating milestones accurately is extremely difficult if not impossible.

3.2.2.2 RAD

RAD (Rapid Application Development) was developed in 1991 with emphasis on creating early prototypes and using iterative development (Mochal, 2001). It works most effectively with small projects and permits a great deal of flexibility for the developer. The negative effects of using this methodology are that generally, when rapid prototypes are developed, not much attention is given to building the documentation; some

iterations provide very little improvement over their predecessor and important issues such as usability may be ignored.

3.2.2.3 USDP

UDSP (Unified Software Development Process) follows the principles of agile development, similarly to RAD. It is Architecture centric whilst being both iterative and incremental. The lifecycle is partitioned into a phases, where each phase can contain several iterations which may overlap; each project milestone is met after each of the iterations in that phase has been completed (Jacobson et al, 1999). Critics of this method argue that the system may not work effectively on projects with a strict timeline or budget.

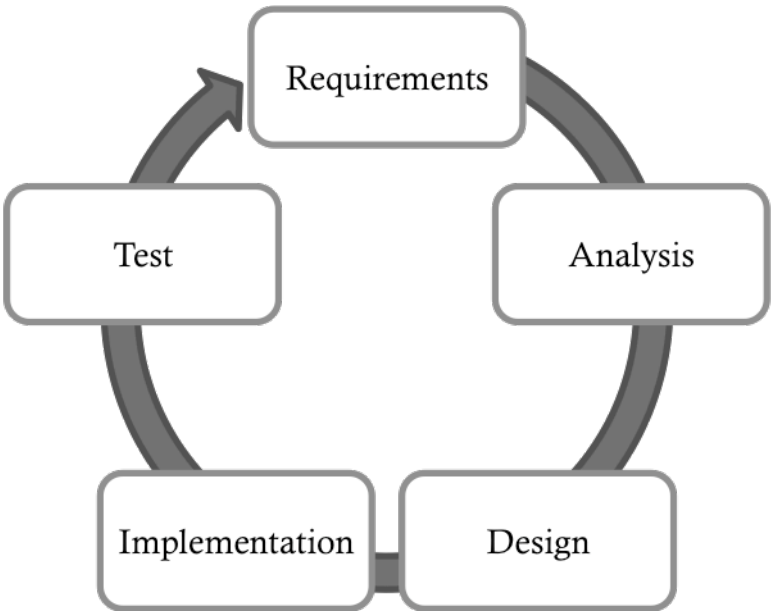


Figure 3.2 - The iteration workflow in USDP

3.2.3 Chosen Methodology

The Waterfall methodology is well suited to this project because of the staggered deadlines that would be imposed throughout the course of development, however it is highly criticised for the implementation stage so forming a hybrid with a simple, flexible, agile methodology such as RAD might be more appropriate for this project.

3.3 Control Interface

It is recurrently documented that perhaps the most important component in an educational game is that the experience mimics or allows the virtual equivalent of a real-world experience as closely as possible (Cakmakci et al, 2003; Liarokapis, 2005; Galarneau, 2005; Xin et al, 2007). Combining intuitive tangible controls with rich media content will help players become physically immersed in the gameplay and that will, in turn, help to create a very effective learning environment (Xin et al, 2007). For these reasons, when practical, a guitar peripheral would be the most appropriate control for the game.

The two primary guitar input options available are analogue and digital signal. Analogue would consist of either an electric or acoustic guitar, recorded via a microphone or line-in directly into a computer, whereas the digital signal could be produced by a MIDI-Guitar, or a standard electric guitar with MIDI Pick-ups. Both approaches shall now be discussed in more detail.

3.3.1 Analogue Input

In order to process an analogue signal into binary data, it needs to be handled by a Digital Signal Processor. Initially the signal must be measured or filtered then converted from its analogue form into digital sequence of data using a convertor. In the case of the electric guitar, its electrical impulses are recorded and are then sampled at a particular sample rate. The higher the rate, the more processor power required, but also the higher the frequencies that can be stored. Each sample has a maximum frequency that can represent it mathematically called the Nyquist Frequency, although human perception is lower than this. Fast Fourier Transforms can be then applied to the samples to approximate the frequencies in the original signal.

This sampling approach presents some problems because of the intrinsic ambiguities inherent to stringed instruments. Note recognition can become an arduous task when the guitar is played very quickly, this could be due to overlapping notes, bends, harmonics or simply because of an insufficiently powerful processing system (Cabral et al. 2001). It is also exceedingly difficult to correctly identify groups of notes such as chords because of the many spectral components that compose the sound.

3.3.2 Digital Input

When a digital signal is used by the system, much less processing is required because the data being transmitted already identifies what notes are being played. MIDI is the industry standard recording interface

in music, it can be a very useful tool for working with instruments and replicating what is played. The data does not contain any information about the waveform, rather a series of commands for the machine interpreting it (Sheppard, 2003). The negative effect of using this system is that MIDI is unable to compute certain expressive techniques such as harmonics.

3.3.3 Primary Control

MIDI was chosen for interfacing the guitar with the game because of the level of difficulty and number of restrictions imposed through processing an analogue signal opposed to using digital data.

There are several methods for transmitting a MIDI signal from a guitar to a computer. The most reliable options are using a MIDI-Guitar such as the Yamaha EZ-AG or the Casio MG510. When the EZ-AG was introduced by Yamaha they marketed it as a learning toy, this deterred intermediate players and as a result the instrument and is now unavailable to buy and difficult to locate; whilst the Casio was received very well by learners and professionals and as a result is still a highly sought-after instrument even though its circuitry is almost 20 years old.

The alternative approach to using a guitar with built-in MIDI support is to use a MIDI Interface. This can interpret a signal and convert it into MIDI data either via a microphone or by connecting directly to a divided-pickup. A common misconception is that the pickup alone delivers the MIDI-Signal, whereas in fact its main purpose is the split the signal into separate channels before it is processed.

3.3.3.1 Control Requirements

In order to maximise learning potential through maintaining flow and immersion in the game, it is essential that the controller feels as synonymous to a guitar as possible. For this to occur, the following details must be addressed –

- *Physical likeness*, the hardware must resemble a six-stringed instrument. A guitar shaped peripheral would be most suitable to help to create an authentic learning environment (Galarneau, 2005)
- *Latency*, the controller must have a fast and accurate response, the human ear is very sensitive (Rothstein, 1992) and Table 3.1 illustrates the effects of latency vs human perception. If lag time is unavoidable, then consistent latency is very important to maintain the game's playability.
- *Standards compliance*, the peripheral must conform to MIDI standards to maximise the game's accessibility and allow future iterations to support a wider range of controllers.

Table 3.1 - MIDI Latency vs Human Perception (Sheppard, 2003)

Latency	Human Perception
<75ms	Seems instantaneous
100ms	Short delay appears with fast notes
150ms	More noticeable delay
>200ms	Delay is very noticeable

Initially the Yamaha EZ-AG was chosen, but early feedback indicated the player was too easily removed from an immersive experience because they were very aware that they were not playing a genuine guitar and therefore felt less motivated to play a learning game. When compared to a standard electric guitar it was also observed that absolute beginners were noticeably struggling to strike the correct strings, whilst intermediate players felt restricted by the binary nature of the fretboard and did not appreciate being unable to include the subtle nuances that define their playing style.

After researching an alternative solution, a Fender Stratocaster with the Roland GI-10 Interface and Roland GK-2A Divided pickup were chosen. The effect of this change was marked in the reaction of the users; both the beginners and intermediate players were much happier playing with a real guitar although the beginners agreed that it would be even more improved if the Stratocaster had lighted frets.

3.3.4 Secondary Control

A selection of the Mini Games, where the guitar control method isn't used will require a separate control interface. Several options will now be considered and discussed.

- *Keyboard*, a common choice, but often entails memorisation and pause for recall, temporarily interrupting the user from the flow of playing.
- *Mouse*, the ubiquitous pointing device, very effective at pointing at to objects in arbitrary positions, with a well designed system, it can be as fast as finger pointing. Theories such as Fitt's Law can be adhered to for designing the optimum position and size of hit targets.
- *Joystick*, a poor general cursor control device that is only effective in applications such as flight simulators. Unlikely to be natively supported with the language chosen.

3.3.5 Mobile Control

The joystick or arrow keys built into the phone will control the mobile game; touch screen phones will not be tested in this project.

3.4 Language and Platform

The requirements for a suitable programming language for this project are as follows:

- *Well Documented* – If the language is popular then it is likely that there is plenty of learning material and examples to learn from, speeding up the design process and helping eliminate major pauses in construction due to errors or limitations.
- *Extendible* – Due to the nature of the project, it should be simple to modify or add to the code throughout various stages of the prototyping process.
- *XML Support* – One of the main aims of the project was to facilitate the use of external data.
- *Efficient* – The language chosen should not be too demanding on users system resources and should be able to process MIDI messages with minimal latency.
- *Portable* – It would be preferable for the prototypes produced to be capable of running on multiple platforms.
- *Graphics Libraries* – The prototypes produced should have an effective GUI (Graphical User Interface).

3.4.1 Java

Java would be the natural choice for this project as I have learnt to write and manipulate advanced code with it throughout my degree and it supports all of the requirements. It is excellent as a multi-platform solution and there are plenty of related classes to learn from. It is slightly hindered by its graphics capabilities and animation libraries, whilst it does offer the Swing library for creating GUIs, this solution is not preferable for rapid prototyping.

3.4.2 ActionScript 3

ActionScript has developed greatly in recent years, and its installed on an abundance of computers (according to Adobe 98.8% of users in mature markets such as the UK, USA and Canada have access to

Flash Content). It has native support for XML parsing, a low memory footprint and full integration with Adobe Flash.

3.4.3 C++

C++ is a very powerful mid-level language that has been commonly used for making audio and MIDI applications in the past. It is extremely efficient and has mature graphics libraries; yet I have little development experience with it and as the project is aimed at rapid prototyping it may not be the best option in this case.

3.4.4 Chosen Language

ActionScript has been chosen for the main development language in this project. The motivation for this is based on the relative ease of prototype development and lack of guitar-learning games created with ActionScript. However, ActionScript is sandboxed and prevented from accessing many of the processes that the game will require. Adobe AIR alleviates some of these restrictions as it provides certain file access permissions, however it will still require separate software to communicate with the MIDI guitar. Java has been chosen for this task, primarily because I can apply knowledge and experience gained from my degree. Whilst there are also many Java applications that use MIDI libraries freely available on the Internet, there are very few, if any, that process a MIDI Signal to extend ActionScript's capabilities.

3.4.5 Chosen Platform

The development environment used for the games will be Mac OS X 10.5. It is likely that the majority of games produced will be able to run on either Windows or Linux too with little or no modification, provided that the user installs the appropriate drivers beforehand. Time permitting, this may be tested in the post development phase.

3.5 System Data Architecture

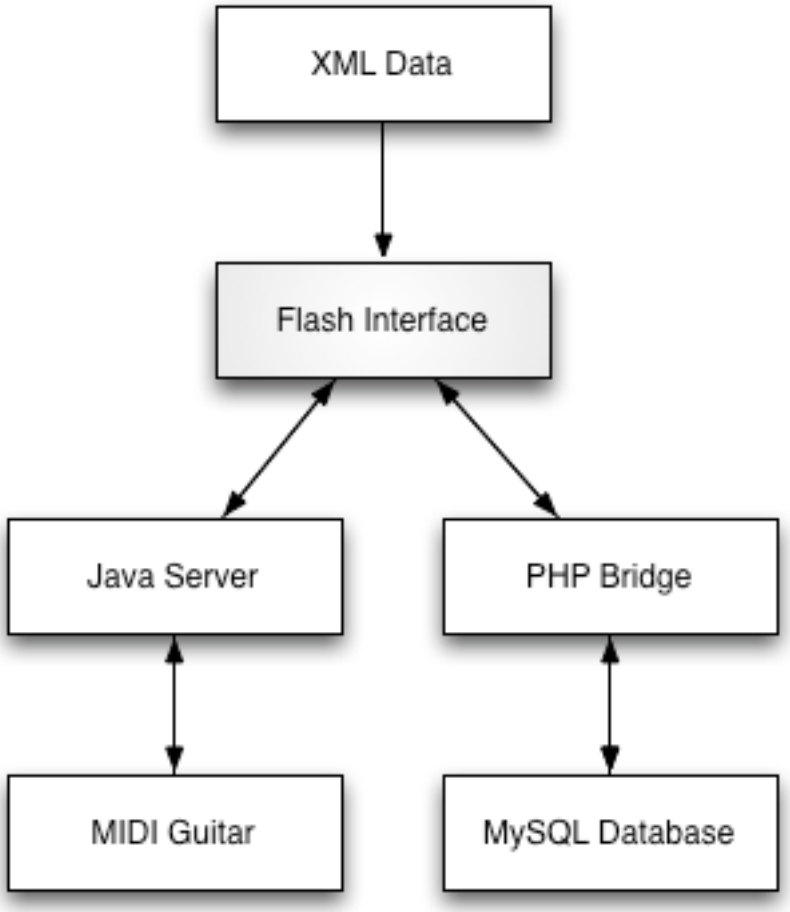


Figure 3.3 – Diagram of the proposed data workflow

Figure 3.3 illustrates the data architecture of the project; the ActionScript based Flash interface will read XML data for tasks such as selecting chords or matching the corresponding MIDI signal to a note string. The Java server will interact with the Flash interface via a binary socket connection, once the appropriate system drivers have been installed on the users system their guitar will be able to connect to the Flash Interface via the Java Server.

3.6 User Interface

How information is presented by the game is very important, the aim of a musical teaching game is transfer appropriate knowledge in an effective manner (Liarokapis, 2005). Roads (1996) concluded that the most direct way of learning how to play a musical instrument is by observing a teacher playing it. This coupled

with personal frustrations experienced with standard written material led to the concept of combining necessary written information with video when possible. Xin et al (2007) discuss 'Mixed Reality', where virtual entities are added on top of live video to merge physical and virtual worlds. Whilst this idea is slightly different to the aims of the project, there is no reason why it could not be alluded to for the purposes of enhancing the user's sense of immersion.

Visual coherence is achieved by providing virtual entities physical handles that the user can manipulate (Xin et al, 2007). The most important requirements are to realistically overlay computer-generated information and allow the system to be controlled with real time performance (Liarokapis, 2005). This can be translated to playing the guitar along with a video by having the guitar control the progress of the video; i.e. if the correct notes are played synchronously with the song then the video can progress, otherwise it will pause until the user plays the correct notes.

As described in Section 2.3 of this report, the game should also try to adhere the guidelines devised by Prensky (2001) and Jones (1998). The fact that users have a preference for graphics over text and a desire to be constantly aware of their score or progress are likely to be the most important issues to address.

3.6.1 Experiments

The mini games and experiments will all serve a specific purpose and their visual design will not be considered unless it is related to the aim of the task.

3.6.2 Mobile Game

The mobile game will have a very simple and recognisable guitar neck style interface. Graphical requirements will be minimised in order to allow the game to be played on many devices and reduce battery strain.

Chapter 4. Implementation and Testing

This chapter presents a commentary on the development of the game features; some of the obstacles that were faced and the decisions that were made to overcome them.

4.1 External Data

4.1.1 XML

In accordance with the aims of the project, XML will be used to provide external data to the game, this will not only include the song data but also the system's knowledge, for example, the data required to recognise a chord pattern. XML is frequently used for storing data in software and it is well supported by ActionScript 3.0, which utilises ECMAscript for XML (Section 3.4.2). Allowing the software to cast the data into a native type that is similar to Arrays.

4.1.1.1 Song Data

Initially the XML structure below was chosen, in an effort to keep the format simple and human readable, fret numbers were used to identify the notes that are to be played. The reasoning behind this decision was to allow the user to familiarise themselves with it quickly because of its resemblance to the popular ASCII Tab format; whilst trying to inject metadata and impose some basic standardisation, such as constant letter spacing for gauging timing and title tags.

```
<?xml version="1.0" encoding="UTF-8"?>
<songs>
<song>
<title>C Minor Pentatonic</title>
<artist>C Minor Pentatonic</artist>
<urltitle>c_min_pen</urltitle>
<tab>
-- -- -- -- -- -- -- -- -- -- 08 11 08 -- -- -- -- -- -- -- -- -- --
-- -- -- -- -- -- -- -- 08 11 -- -- -- 11 08 -- -- -- -- -- -- -- --
-- -- -- -- -- -- 08 10 -- -- -- -- -- -- -- 10 08 -- -- -- -- -- --
-- -- -- -- 08 10 -- -- -- -- -- -- -- -- -- -- -- 10 08 -- -- -- --
-- -- 08 10 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- 10 08 -- --
02 03 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- 11 08
</tab>
</song>
```

To accompany this song data, an additional notes.xml file was required.

```
<!--String 6-->
<note name="E" octave="3" midiValue="40" stavePos="0" position="s6f0" />
<note name="F" octave="3" midiValue="41" stavePos="1" position="s6f1" />
<note name="F" octave="3" midiValue="42" stavePos="2" position="s6f1" pitch="#"/>
```

Whilst this system met its requirements, it was too inflexible and imprecise. For the software to be capable of scrolling through the represented graphical data synchronously with a song more accurate timing information needed to be provided in the dataset. The successor to this approach was to use Adobe's proprietary video format Flash Video (.flv), which was chosen because it supports cuepoints.

4.1.2 Cuepoints

A cuepoint is an event that is triggered at set intervals during a video's playback. They can be embedded into the video file or created by ActionScript at runtime, for this project the preferred method would be to dynamically create the cuepoints. This would enable infinite sets of data to be applied to the same video, which could then be swapped throughout the video's playback, e.g. if the level of difficulty were to change.

Regrettably, after rigourously testing dynamic cuepoint generation it was apparent that the accuracy of the cuepoints' positions was too inconsistent, and unacceptable latency of up to one second could frequently be observed. Therefore the alternative method of hard coding the data into the video file was adopted.

Several tools exist for embedding cuepoints into an flv, FLVTool was a prime candidate because it could manipulate the video files via the command-line and was theoretically accessible by ActionScript meaning that a GUI (Graphical User Interface) could be implemented for creating songs. However, it imposed restrictions on the format of the data accepted, specifically the accuracy of the timestamps. Eventually Adobe's own tool, Flash Video Encoder was chosen, this parses XML files that adhere to the following format.

```
<?xml version="1.0" encoding="UTF-8"?>
<FLVCoreCuePoints>

<CuePoint>
  <Time>0</Time>
  <Type>navigation</Type>
  <Name>video_info</Name>
  <Parameters>
    <Parameter>
```

```
    <Name>Title</Name>
    <Value>A Test Video</Value>
    </Parameter>
    <Parameter>
    <Name>Artist</Name>
    <Value>A Test Artist</Value>
    </Parameter>
    <Parameter>
    <Name>Tuning</Name>
    <Value>40,45,50,55,59,64</Value>
    </Parameter>
  </Parameters>
</CuePoint>

<CuePoint>
  <Time>1411</Time>
  <Type>event</Type>
  <Name>cp1</Name>
  <Parameters>
    <Parameter>
    <Name>Notes</Name>
    <Value>01,00,03,04,00,00</Value>
    </Parameter>
  </Parameters>
</CuePoint>
```

The initial CuePoint node contains the meta-data for the song, this approach is not officially supported or documented by Adobe, but because of the strict layout rules imposed by the parser it was an effective solution in this case. The tuning parameter refers to the MIDI code that would be generated by each string if it were played 'open', i.e. no strings were pressed down.

The second CuePoint node represents the first action performed on the guitar track. The two important child nodes are *Time*, which represents the time from the start of the video in milliseconds (ms), and *Value*, which describes the fret positions on each of the six strings.

Whilst this new file is still technically human-readable, when compared to the pseudo-tab XML it is very convoluted and would require parsing before being read, or a GUI if it were to be written by a human.

```
<CuePoint>
       <Time>2000</Time>
       <Type>navigation</Type>
       <Name>CuePoint1</Name>
       <Parameters>
              <Parameter>
                     <Name>CuePoint1</Name>
                     <Value>Introduction</Value>
              </Parameter>
       </Parameters>
</CuePoint>
```

4.1.2.1 Parsing CuePoints

Once Actionscript accesses the Tuning node, a two-dimensional array of fret values is created. This is done using the function setFrets shown below.

```
private function setFrets(noteArray:Array):Array
{
  var frets = new Array();

  noteArray.reverse();

  for (var i:uint = 0; i < noteArray.length; i++) {

    frets[i] = new Array();

    for (var j:int = 0; j < NUM_FRETS; j++) {
      frets[i][noteArray[i]+j] = j;
    }

  }

  return frets;
}
```

The constant NUM_FRETS is typically defined as 22, this relates to the number of frets found on the neck of the Fender Stratocaster Guitar. For standard tuning (EADGBe), the returned array would be

```
//First String
frets[1][80] = 16
frets[1][81] = 17
frets[1][82] = 18
frets[1][83] = 19
frets[1][84] = 20
frets[1][85] = 21
//Second String
frets[2][59] = 0
frets[2][60] = 1
frets[2][61] = 2
```

frets[1][80] refers to the MIDI Note 80 found on the 1st string. Due of the nature of Arrays, typically the first index would be 0 but to keep the design closely mapped to the fretboard, frets[0] is nullified in this software. frets[2][59] refers to the 0th fret which in this case is the 2nd string played open i.e. the string is not pressed down when hit.

After creating the custom Event class com.axeclass.events.AxeClassEvent it is possible for an event to be dispatched every time there is an action performed on the guitar. When a listener receives this event actions

can be performed to check if the user is playing the correct notes, this is achieved by coupling the Flash client with a custom Java server.

4.2 Java Server

4.2.1 MIDI Connectivity

To transmit the incoming MIDI signals to Flash for processing, a java server program will run in the background.

Initially the Open Source Flash Server Red5 was chosen as the bridge between the MIDI input and Flash. It is a Java based streaming server application that supports MIDI and used RTMP to deliver binary data to Flash. It required a reply message from the Actionscript client after sending the data, similarly to a three-way handshake. This added unnecessary overheads and the server itself was slow and overly complex for the task of delivering MIDI messages. This resulted in a custom Socket Server being implemented.

4.2.1.1 Socket Connection

Whenever the program receives a MIDI signal it will write the output to a buffer and flush it to a socket that an Actionscript client will be listening to. The socket is an end-point of a communication link between two programs running over a network (Technion, 1997), in this case the network will be localhost. The Socket classes Socket and ServerSocket are used to represent the connection between the client and server program respectively.

There were two types of socket connection to choose from, Datagram Sockets and Server Sockets. Datagram sockets use UDP, which is a connectionless protocol, meaning that delivery is not guaranteed and that every packet sent needs to have extra information about the socket descriptor and the recipient address (Reilly, 1998). Sever Sockets use TCP and in order for communication to take place, the connection must be present between both sockets. Once the connection is made, data can be transferred in either direction with a guarantee of data delivery. The two options could be compared to the postal and telephone service respectively. A Server Socket connection was chosen for this project, as it is very important that all of the input data can be processed by the game for the scoring to function correctly. In an effort to lower bandwidth overheads the data transmitted will only contain pertinent information and an effort will be made to keep the messages short and succinct.

Music games are inherently time critical, once a signal is received by the system it is crucial that it can be processed as quickly as possible. For testing purposes the game used a constant port and string value for the MIDI device connected, although it would be possible to create an admin interface to make these dynamic in case there were firewall restrictions or the user was not playing through a Yamaha UX-16 USB to MIDI cable.

```
server = new ServerSocket( 5555 );
```

After the server has been instantiated, the receiver can then listen for input events and write them to the socket.

```
strMessage = "0,"+message.getData1(); //note off

strMessage = "1," + message.getData1(); //note on

strMessage = "2," + get14bitValue(message.getData1(), message.getData2());//bend

strMessage = strChannel + strMessage;
```

In this case strChannel represents the MIDI-Channel, as the setup used supports multiple channels, each string was assigned its own individual channel making it much more efficient for the software to determine exactly which fret position was played. Without this feature more calculation would be required from the software due to the recurrence of notes in various positions on the fretboard.

Table 4.1 - The Proposed Java-Flash Communication Message Format

	Action ID:int (0-2)	String ID:int (1-6)	Note/Value:int (0-127)
Note-On	1	3	52
Note-Off	0	3	52
Pitch-Change	2	3	12

```
var message:String = ‘1,3,52’;
```

Table 4.1 illustrates the proposed message format for each possible packet sent from the Java server. Once AS3 receives the message it can split the values into an array of three nodes as below

```
Split_message[0] = 1;
Split_message[1] = 3;
Split_message[2] = 52;
```

Assuming that no other notes were sounding, the array would then be set to the values below.

```
Var Strings:Array = -1,-1,52,-1,-1,-1;
```

In this case the third node of the Strings array represents the MIDI Note 52, because the system has learned the notes from the XML files it imported it can inform the user that the note played was E

```
<octave num="3">
…
<note midi="52" letter="E" accidental="" />
```

If the current cuepoint in the song were equal to this then a score would be registered. Once the player mutes the note himself or it mutes over time, the note-off message will be sent from Java and the array will revert back to negative integers.

```
Split_message[0] = 0;
Split_message[1] = 3;
Split_message[2] = 52;

Var Strings:Array = -1,-1,-1,-1,-1,-1;
```

```
//note/value
if action < 2
midi value
else

bend velocity
```

4.2.2 System Exclusive Messages and Fret Lighting

One of the experiments' purposes was to determine if it was possible to light predetermined sequences of frets on the EZ-AG's neck via software and the MIDI-USB cable, a consequence of this would allow the user to have the option of setting the game to light the correct positions for the song on the fretboard. The first six fret positions on the EZ-AG's neck are backed with LEDs so that when the fret button is pressed it lights up during play. Applying this functionality would further enhance the Mixed Reality paradigm (Xin, 2007) and would contribute greatly to the sense of immersion in the game.

The EZ-AG manual (Appendix D) specifies the format of the message used by the guitar. With this information some code samples from jsresources.com could be modified allowing custom SysEx messages to be transmitted to the guitar.

Table 4.2 – An Example SysEx Message for lighting a fret position on the EZ-AG

1	2	3	4	5	6	7	8	9
Message Header		Device ID	Model ID	Delivery Method	Function ID (ll)	String Number (mm)	Note Number (nn)	Message Footer
F0	43	7F	00	00	03	05	2A	F7

Table 4.2 represents a message that instructs the EZ-AG to light the second fret on the sixth string. Parts 1, 2 and 9 signal the start and end of the message respectively. These remain constant alongside parts 3 and 4. Part 6 is the unique function id and in this case parts 7 and 8 denote the exact position on the fretboard that is to be targeted.

Unfortunately this system had a couple of limitations. The main concern was that the guitar does not support lighting of the 0^{th} fret i.e. the string. This is a major flaw as playing the correct strings is equally as important as having the correct finger positions on the fretboard. To overcome this problem, when a message is sent to the guitar it instructs it to light the sixth fret (highest possible backlit position) for each string that should be played. This solution is not ideal because the sixth fret cannot be lit as a finger position instruction, an alternative to this method would be to blink the sixth fret's light by sending continuous on-off signals, however, this also caused problems in fast songs where the positions changed rapidly.

Due to the problems that were faced and the lack of backlights on all twelve frets, the functionality was not incorporated into the game as it may cause confusion for the beginner players. However, the system was very effective at demonstrating simple chord positions such as the Major deviations (Figure 4.1) as they rarely involve fret five or above. The software also successfully changed other parameters on the guitar during this experiment, including volume and tuning. These options were going to be included into the final game until it was decided that a real electric guitar would be used instead of the EZ-AG, making their inclusion redundant.

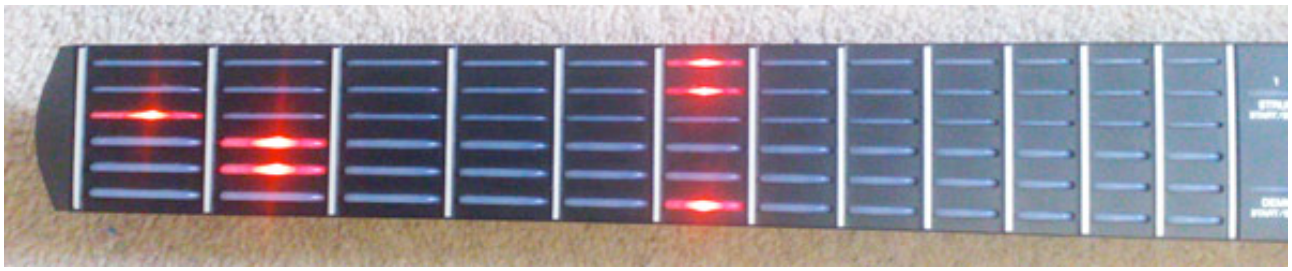


Figure 4.1 – EZ-AG's lighted frets displaying an 'E' Chord

4.2.3 Chord Recognition

One of the motivations for choosing MIDI over a conventional guitar and microphone setup was for the possibility of developing fast and accurate chord recognition. As explained in Section 3.3.1, this would be very difficult to achieve through digitally sampling the sound produced by an acoustic or electric guitar because of all the intricate variables that are introduced when playing a stringed instrument. This issue is not as prevalent when using MIDI equipment, thus making it possible to produce a chord recognition system in the experiment stage of the project.

To achieve chord recognition the system must first acquire the knowledge of what data signifies a chord. All of this is stored in an XML file, which is then loaded into the game and casted into array of key-value pairs.

```
<chord name="A" MIDIvals=",0,2,2,2,0" FINGERvals=",,1,2,3," sound="a" level="1" />
<chord name="A min" MIDIvals=",0,2,2,1,0" FINGERvals=",,2,3,1," sound="am" level="2" />
<chord name="A min 7" MIDIvals=",0,2,0,1,0" FINGERvals=",,2,,1," sound="am7" level="3" />
```

MIDIvals refers to the required MIDI values for each string, FINGERvals indicates the recommended finger positions for the chord, sound is the filename of the mp3 file stored in the software's directory and level acts as a guide for the perceived difficulty of the chord, i.e. an advanced player would be able to play level <= 3.

When a guitar activity event is triggered, one of the processes is to check the current status of each of the six strings against the MIDIvals data for each chord that has been loaded into memory. If the sequence of values matches one of the chord values then the system can inform the user the name of the chord that they are playing.

4.2.3.1 Emulating Note Decay

The Chord Recognition system described was flawed with the EZ –AG as it does not produce a note-off signal when the note's 'sound' has completely faded out; rather the instrument will provide the instruction once the fret position is released, or in the case of a stricken open string, no data will be transmitted.

To overcome this, a technique needed to be devised to realistically imitate note decay. It was important that there was a sufficient buffer time for the chord to be played in as each note signal is sent individually i.e. there may be a slight delay between each string that is struck. This led to the system shown in Figure 4.2 being developed.

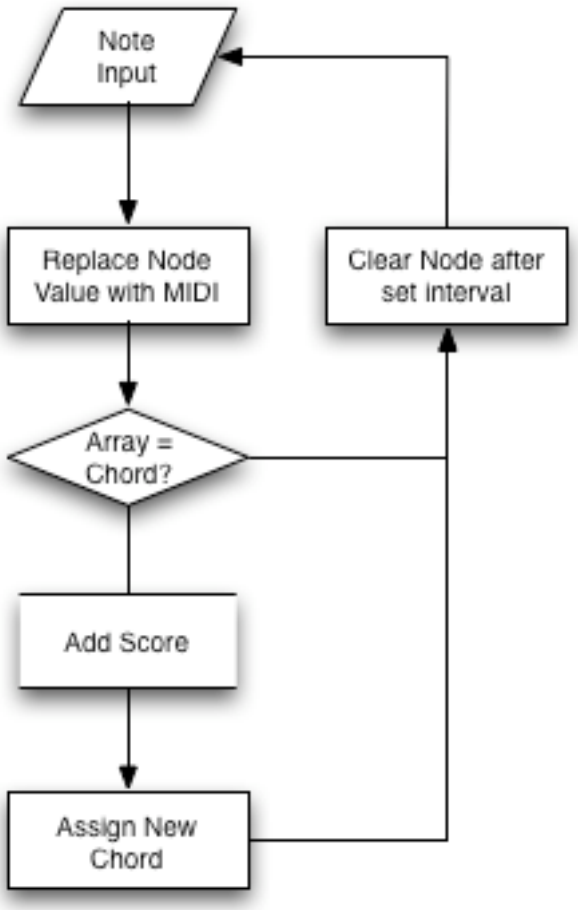


Figure 4.2 – Flow Chart for Chord Recognition

Firstly a fixed-length array of six indexes was created as the guitar utilises six channels to transmit and receive the MIDI signals. The first index of the array would hold the first string's value etc. From this point on, any new MIDI signal, which was received, would immediately be filtered by Flash into the corresponding array index. After which, Flash would concatenate each value in the array into a single string and compare this against the string value of the current chord. If the strings matched then the system would increase the score and introduce a new chord to the user, if the strings were not equal then no score would be added and the user would have to attempt the chord again. In both cases a timer would now be started specifically for this string, if one already existed then it would be overwritten. Once the timer finished then the value that corresponded to this string would be nullified.

Different delay durations were tested during gameplay and 0.6 seconds was eventually chosen, although this amount of time is less than the 'average' length of the note decay, it works well in this context as the majority of users analysed were frustrated if they had to wait until the guitar fell silent before they could attempt another chord.

To improve this system, the delay could vary depending on the velocity at which the string was struck. The greater the velocity the longer the delay as the pickups would usually produce a signal for as long as the string is vibrating inside their magnetic fields.

4.3 Experiment Results and Main Game Design

Rapid prototypes and experiments (Figure 4.13) were developed in the initial stages of the project before the game's design was finalised. This was decided upon after discovering that during a game's build, developers are regularly confronted with unpredictable design obstacles or restrictions (Sloper, 2003) that make them redesign their initial conception.

Several aspects of the game will now be discussed with focus placed on describing features that were developed as a result of building the prototypes.

4.3.1 Scoring

A score is registered when the user plays the correct note in time with the information being delivered by the game. The percentage is then formulated from the current score and the total possible current score. Doing

this allows the user to have constant visual notification for how they are progressing, one of Jones' (1998) recommendations (Section 2.3.1).

If the guitar does not provide the correct MIDI sequence at a cuepoint interval then the game pauses until they do, whilst this is happening a counter increments in the background, the accumulated count at the end of the game could then be deducted from the score. When the counter was visible it appeared to hinder learning as users experienced stress from seeing the counting clock, this is compliant with Clark's (2006) assessment that too many game controls can hinder intrinsic motivation. The system could be improved by providing the option to disable the game pausing method and taking into account the number of times the strings are struck to calculate an accuracy percentage.

4.3.2 Virtual Fretboard

An on-screen fretboard was created early in the project's development and used in sundry experiments. Its objective was to assist in providing on-screen feedback for the user and encourage beginners to look away from their hands and begin to feel their way around the neck of the guitar.

When the system receives a note-on message for a fret position it maps it to the fretboard graphic and highlights that fret on screen. This could be extended in many ways, for example it could be used as an alternative to physical fret lighting (Section System Exclusive Messages and Fret Lighting4.2.2). By showing the fret positions on screen rather than on the players guitar neck it may also help them to develop their 'mirroring' skills, i.e. the ability to replicate a performance whilst watching somebody else playing it.

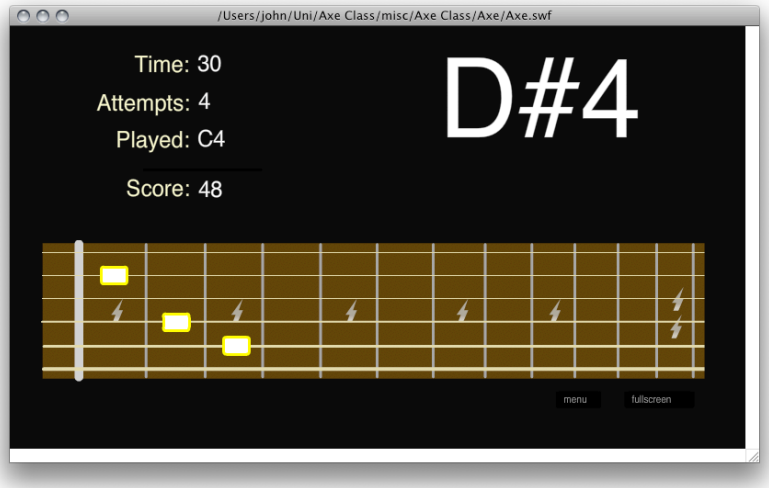


Figure 4.3 - Virtual Fretboard

4.3.3 Presenting Data

Many different methods were tested before deciding on which would be used for displaying the song data in the game. Initially, inspired by resources such as Wilson's Guitar Games (http://guitargames.net), it was hoped that there might be a unique delivery style that could be proved to be more effective than the traditional methods. Several abstract concepts were explored and developed in the experiments, some of them served very specific purposes whilst others were more generalised.

4.3.3.1 Minigame Concepts

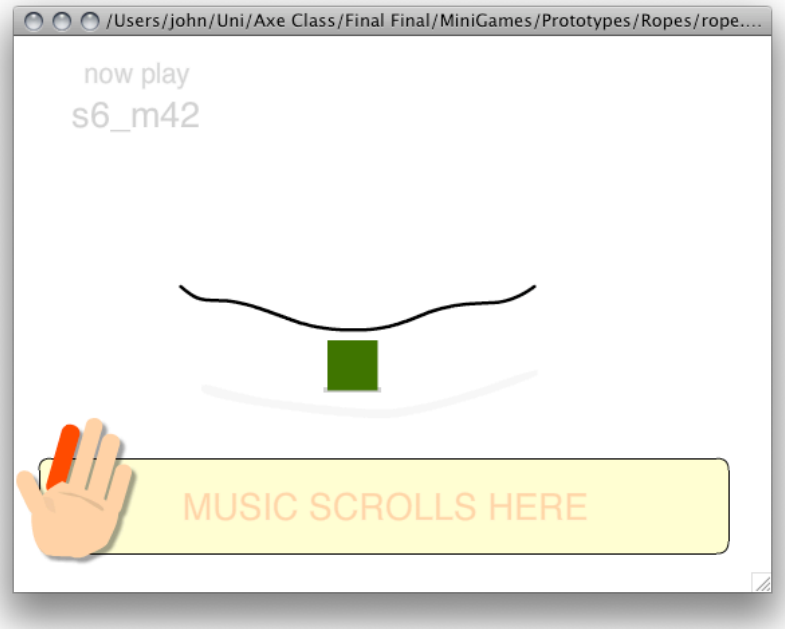


Figure 4.4 - Rope Jump Game

The purpose of the rope jump game (above) was to help the player develop their sense of timing. It acts as a metronome, instructing the user to play the note indicated so that their avatar could jump over the rope at the right time. The finger position in the XML instructed the game which finger should be highlighted; this was to assist beginners on using the optimum hand position when playing sequences such as scales. The idea was abandoned after discovering that there was no way of implementing a very precise timing mechanism on multiple systems.

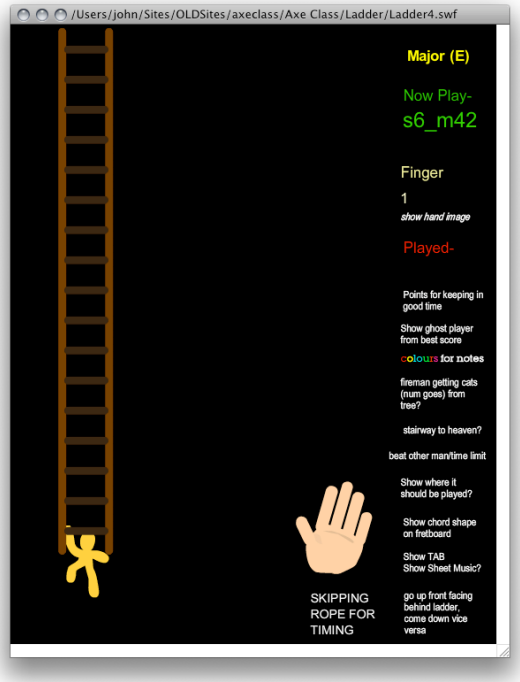


Figure 4.5 - Scale Ladder Game

The Scale Ladder Game was specifically designed for teaching the formation Scales on the guitar. The number of rungs on that ladder is equivalent to the number of notes in the ascending section of the scale. A note from the scale would be displayed and then when the user played it the character would climb another rung. Once the user's climber reached the top of the ladder they would have the play the scale in reverse for him to descend. This could then be used to help the user build dexterity in their fingers and a consistent speed increase by racing their climber against a computer controlled climber.

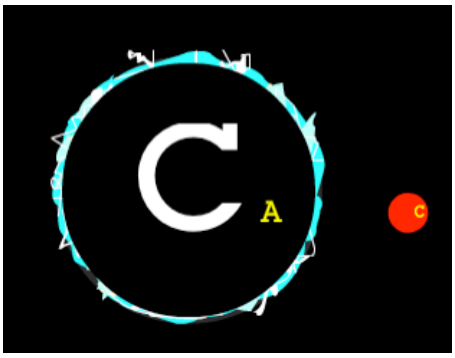


Figure 4.6 - Fly Zapper Game

The Fly Zapper Game was to put a spin on traditional scrolling. Flies would animate towards the centre of the screen from all angles, the premise is that the player needs to 'zap' them as they pass over the zap ring. This would be achieved by playing the corresponding note letter displayed by the fly and/or in the middle of

the ring. This system could also mix up fret positions, notes from the stave or even chords. By using different coloured and sized bugs to represent different instructions this would appeal to the perceived user preference for graphics over text (Prensky, 2001).

4.3.3.2 Scrolling

After reviewing initial feedback, it was decided that the three final display candidates would be based on previous standards set by other games or software –

- Continuously scrolling from top to bottom, inspired by Guitar Hero

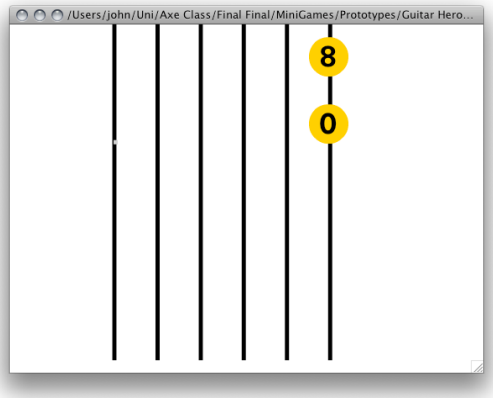


Figure 4.7 - Guitar Hero Style Scrolling

- Continuously scrolling from right to left, akin to a scrolling platform game

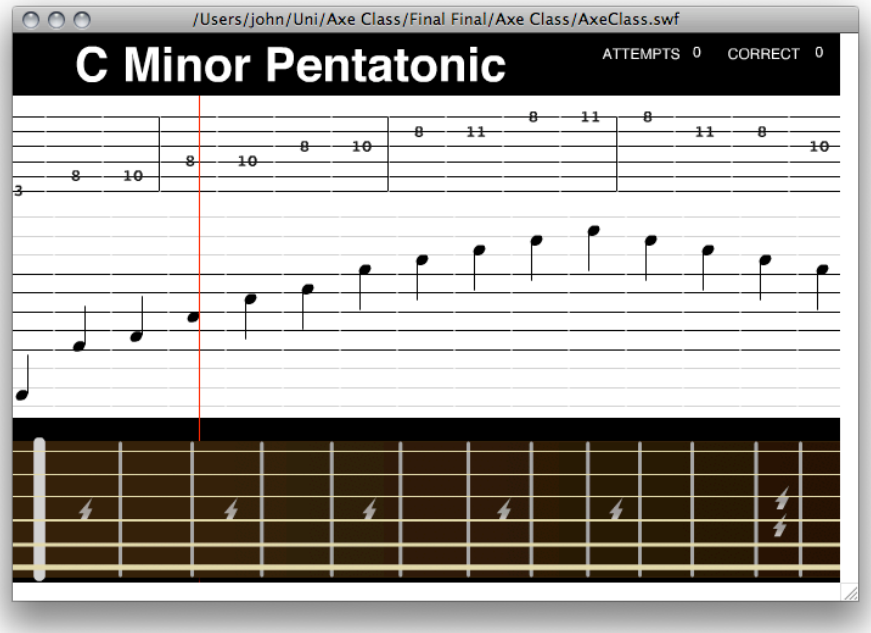


Figure 4.8 - Continuous Scrolling

- Scrolling a reading line from left to right, similar to Guitar Pro

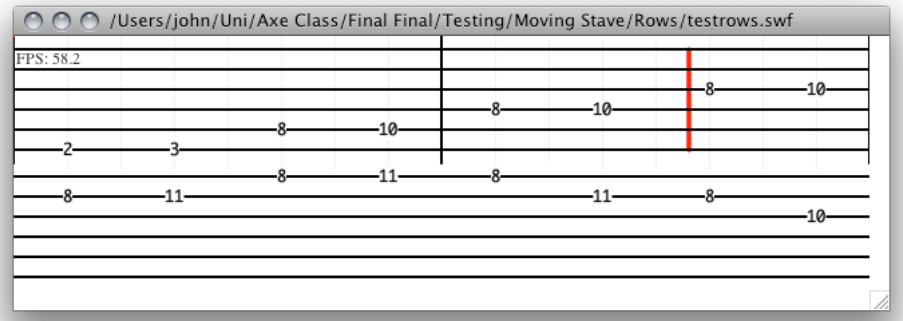


Figure 4.9 - Left to Right Scrolling

Before building the prototypes it was assumed that novice players would be most comfortable and familiar with the Guitar Hero style notation, as it has been embraced as the standard method for presenting information in Rhythm games.

This was quickly disproved as the 90 degree rotation of the lines added an unnecessary level of abstraction and players were much more comfortable with the data scrolling how it would be read on paper. The effectiveness of this approach lies in the simplistic design of the buttons on the controller (Figure 4.10), in this single line fashion it could be argued that the hardware is more reminiscent of a piano keyboard than a guitar. The vertical scrolling method is most suitable to this style of control because the user's horizontal finger positions directly relate to the stave lines displayed.

Figure 4.10 - Guitar Hero Style Controller's Buttons

This outcome meant that the chosen method would either be the continuous scrolling or page style scrolling, where the page shifts up each time the reading line meets the right hand side of the screen. Assumptions were quelled for the second time, by this stage it was thought that users would prefer to have the data as representative to reality as possible and would therefore opt for the moving reading line method. However, the continuous scrolling method was most popular. The reasoning behind this was that paper imposes a restriction of a finite width, and the lines are forced below because of this. With multimedia however there is an unlimited 'width' available so there is no reason to translate this accommodation into the software, as

continuous scrolling is far more fluid and intuitive. A continuous scroll is also much more synonymous with Computer Games, such as the scrolling space shooter games, at this point in the development phase a selection of skins (Figure 4.11) were tested to try to mask the fact that the user was learning to read music. These skins weren't as successful as had been hoped so the idea was discarded.

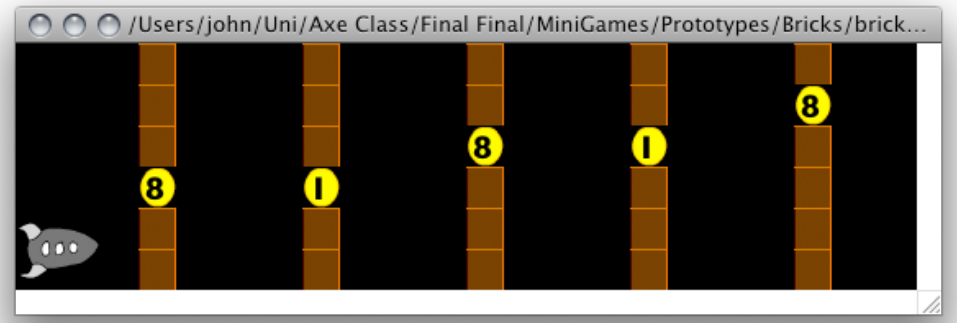


Figure 4.11 - Scrolling tab styled as a space shooter

Originally it was planned that tab notation would scroll with a music stave simultaneously and that either of them could be hidden during the game, this would provide the opportunity for the user to learn both notations. However this introduced some implications, notably on the framerate and timing accuracy of the game so an enhanced Tab was chosen as the sole format. This opened up possibilities for future revisions, such as varying thickness on the tab lines to assist in representing the strings and varying tab box lengths for the note durations.

Figure 4.12 - Final Tab Design

4.3.3.3 Colour Association

Whilst the Guitar Hero style scrolling method was proven to be ineffective, it was felt that there were still lessons that could be learnt from it. It was considered that if the game's elements were similar it might be more approachable to beginners and ultimately help to make the learning process more fun and visually stimulating.

The Guitar Hero Controller (Figure 4.10) uses brightly coloured buttons that are matched by the content displayed in the game, Jensen (2000) suggests that learners invariably perform better when they associate colours with their learning content. This idea could be extended into Axe Class by classifying the strings; this is better than using the frets because there are many of them so the colours would not be as distinguishable.

Unfortunately coloured strings were unavailable to purchase so the strings on the Fender Stratocaster were marked with permanent ink (Figure 4.14). This method was not as effective as was hoped, as the colours were not very prominent, especially on the narrower strings. A more effective solution might be to mark the scratchplate or use different coloured backlights on the Yamaha EZ-AG.

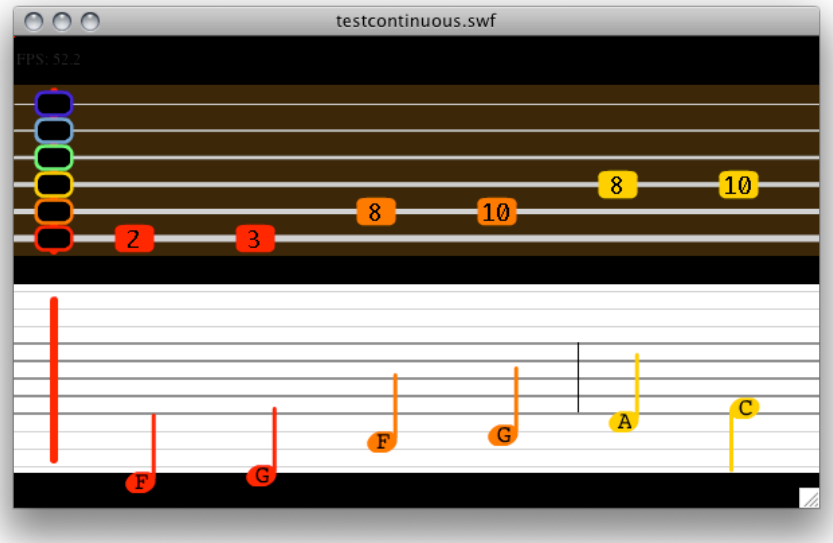


Figure 4.13 – An Early Scrolling Prototype with Colour Coded Strings

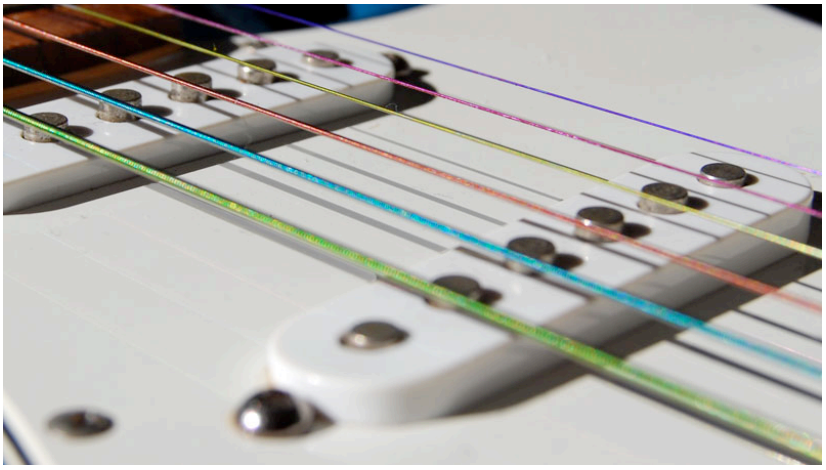


Figure 4.14 – Photograph of Coloured Strings on a Fender Stratocaster

4.3.3.4 X Multimedia Accompaniment

Figure 4.15 - AxeClass Testing Screenshot

The song's video is played behind the scrolling tab (Figure 4.15); it could have different purposes. For a lesson it might just be a person talking with their guitar, or if the user wants to play a song it could be a live performance from the artists who created the song. This is also an effective time-saving approach for developing interesting graphical content for the game (Prensky, 2001). Ideally the source audio would be recorded on separate tracks (stems) so that the guitar track could be muted and then heard whilst the user played a section of the song correctly. Whilst playing stems synchronously was tested in this project, it was not accurate enough when it was applied to an application with video and a large scrolling bitmap. It is also

much more difficult to find a user to find stems for a song that they might want to learn instead of finding the music video or a live performance video. For these reason the video's audio track is used instead.

4.3.3.5 Saving and Loading User Data

Saving and Loading functionality was implemented in the early prototypes of the game, it used a local MySQL database and was accessed via a PHP bridge. To write the data ActionScript would form a HTTP-Request to the PHP file with POST variables and these would be directly translated into the database. For ActionScript to read data it would call the PHP file again but in this scenario the file's MIME type would be set to XML and the content would be served as XML data so that ActionScript could efficiently parse the data using it's native processor. Whilst this solution worked it was not entirely suitable for the project as it would require the user's machine to have a local PHP server with MySQL installed. The alternative would be to bundle this with the game resulting in a superfluous archive, or to use a remote database that would require the user to have an Internet connection whenever they wanted to play game.

Later research lead to the discovery that recently released Flash Player 10 would allow local file permissions and that MySQLite was packaged with AIR, regrettably by this point the project was too near to completion for these to be implemented.

4.3.4 Benchmarking and Memory Management

4.3.4.1 Framerate

Many unforeseen problems arose during testing; they were mainly related to framerate issues. As timing is such a key factor of the game it would be unacceptable to ignore them. To ensure that the prototypes developed would be stable on lower specification machines, measures were taken to ensure that resource usage was kept to a minimum and Kaioa's (2008) AS3 FPSCounter (Frames Per Second Counter) was added to the stage. Early prototypes could achieve a maximum FPS rate of approximately 20; this was reasonable for slow moving animation, however faster sequences displayed noticeable jitter. Through optimising the code, principally by reducing the number of operations that were performed on a repeated basis the average FPS rose to over 50, which is more than acceptable in this case.

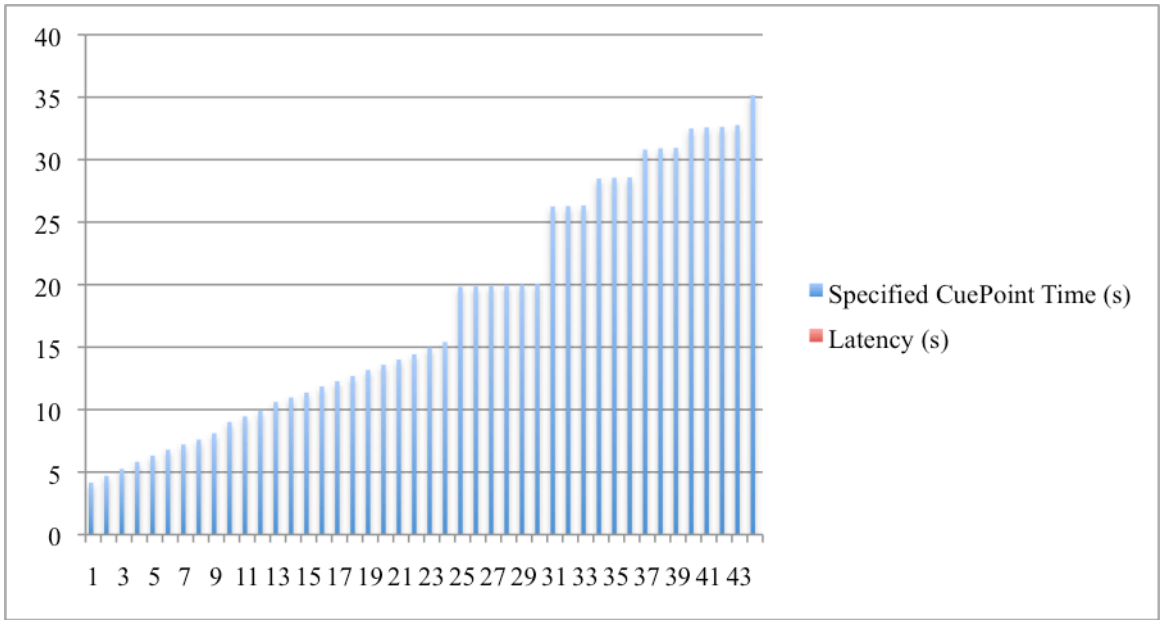
Creating a smoothly animated moving-line stave was much more difficult than was first anticipated. It was always intended to allow the game to be played in a full-screen mode to enhance the immersive experience. However it became apparent that this might make the animation inaccurate on lower specification machines.

The introduction of Flash Player 10 during the build was fortunate because it allows the use of hardware acceleration, which helped things slightly.

The most successful amend for producing smoother animation was to convert the dynamically created tab object into bitmap data. The consequence of this is that animations cannot be performed on the instances i.e. no effects when the user hit that note; this trade-off was acceptable because of the sheer importance of accurate timing in a guitar learning game. Other changes that resulted in the game having a higher Framerate were using the tweening library TweenLite and moving development away from Adobe AIR as this caused a severe drop in performance.

4.3.4.2 CuePoint Latency

To check for the accuracy of the cuepoints timestamps a game was tested at various FPS and an average latency time was recorded. This was done by using a cuepointListener that would output the time stored in the cuepoint and the time that had passed in the movie. It was presumed that there might need to be a constant value that would be need to be removed from the cuepoints timestamp to account for the delay in the recording of the MIDI signal and then the delay in listening and responding to the cuepoint event. However, as is shown in the graphs below the latency was negligible and never accounted for more than 1% of the cuepoint's timestamp. The results were almost identical over a five-minute timespan so only a finite duration is shown.



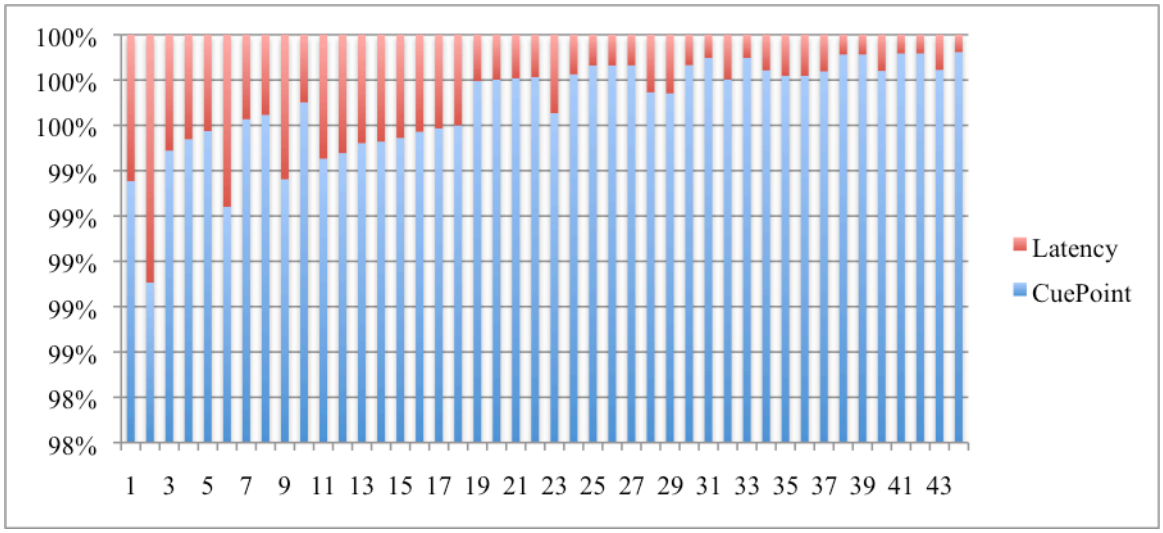


Figure 4.16 - Graphs to Illustrate CuePoint Time vs Latency

4.4 Chord Learning Game

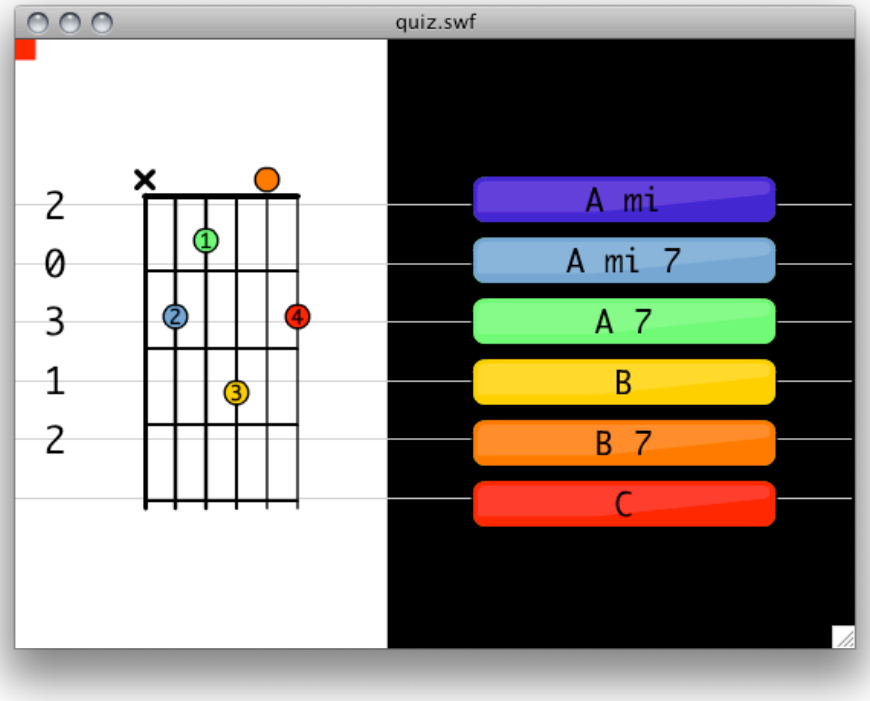


Figure 4.17 - Chord Learning Game

The design of the chord game is very important as it must deliver a lot of information visually and audibly the user. A chord sound is played whilst the tab as well as a chord diagram is displayed on the left of the

screen, the player must then choose which of the answers on the right of the screen corresponds to the chord shown. Over time some of the incorrect answers will disappear and the achievable score for the round will decrease simultaneously. The answers were presented in this fashion for two reasons, firstly so that the game could be adapted so that it was controlled by a guitar incase it was integrated into the main game. This would assist in maintaining the sense of flow whilst playing. Secondly, to use consistent string colours throughout the series to assist learners in responding to instructions autonomously. The data is represented in multiple ways – sound, tab, chord diagram – to provide the possibility of masking certain methods and help the learner become proficient in different types of sight and sound recognition.

4.5 Mobile Game

The mobile game has simple graphics because of the relatively low processing power of mobile technology, yet it should be instantly playable to anyone that can recognise a guitar fretboard. The objective of the game is to learn the positions of notes on the fretboard whilst a guitar is inaccessible to them.

Figure 4.18 - AxeClass Mobile

Section 3.4.2 explained that one of the reasons for using ActionScript was because of the relative ease that the game could be ported to many devices and operating systems. By using Flash Lite a mobile game could be developed. The system was designed to use a simplified scoring system where each fret position is stored in an array and each time they were selected correctly their 'score' value would increment and conversely the score would decrease in the event of an error. By imposing boundary scores of four and zero, four being the best possible score and zero being the worst it was possible to weight the number of times that a fret position was chosen. This allows the system repeatedly choose fret positions that the player needs to practice through the principles of stratified random sampling.

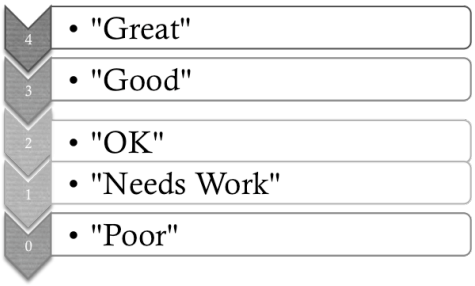


Figure 4.19 - Possible score values for fret positions

The XML file used by the game specifies the 0th fret of each string in the array, so for example the first value in the array contained within attribute values is 7, whilst Array[7]'s value is "E", this denotes that the first string is tuned to "E". This method was chosen as it allows the game to be modified for different tunings.

```
<notes>
	<tuning name="Standard (E)" values="7,0,5,10,2,7" />
</notes>
```

```
["A","A","B","C","C","D","D","E","F","F","G","G"];
```

Measures were taken to ensure that the game could be played in everyday environments. The Flash Mobile Development Suite enables testing multiple conditions, Figure 4.20 demonstrates how the game would be presented with harsh sunlight reflections on the screen.

Figure 4.20 - AxeClass Mobile being tested in harsh sunlight

Chapter 5. Evaluation

This section will critically review and appraise the project to determine whether or not it was successful. The original project aims and objectives will be referenced and discussed, any deviations will be explained and the report's sections will be evaluated.

5.1 Overview

The initial aims of this project were to research pertinent subjects related to teaching the guitar. Subsequently, to build several pieces of software to apply/demonstrate what had been learnt. Although all of my initial goals were achieved to some degree I feel that these aims were not fully met as the task proved to be much larger than was anticipated.

I approached this work with a gung ho, confident attitude assuming that I would be able to build several minigames that interfaced with an electric guitar and to write a report to that end. I now realise that this report cannot do justice to the amount of hours that I have spent in trying to reach the aims that I imposed.

Difficulties arose when I began to try to get an electric guitar to interface with the computer via the Lead-In port. I realised that if I were to attempt this then the majority of my time would be spent trying to master the concepts of signal processing and then managing libraries written in an unfamiliar language such as C#.

After some further research I became influenced by the future recommendations in Bates' paper (2000) where the author concluded that the guitar teaching website which had been tested on students could be greatly improved if it offered more interactivity, especially with a guitar. MIDI connectivity was named specifically. At the time, to my knowledge, no such software had been released or publicised and I was excited as I felt that I was helping to push the boundaries in this field of research.

5.1.1 Research

The Literature Review chapter of this report was split into several sections that I considered to be relevant to the topic. The first section explored the work that had already been undertaken in this field and critically reviewed several games and applications that I tested. I feel that this was one of the most important areas of work as it allowed me to begin to understand how the project would be expected to evolve.

Then various aspects of Game Design were discussed, motivation was a particularly relevant subject as it was found to be one of the fundamental reasons for learners to either give up learning or avoid learning altogether.

To conclude the research I then decided to include a necessary description of the guitar then finish with an explanation of music notation formats. I decided to add the comparison table at the end of the chapter, as I felt that was a good harbinger of my choice to implement Tab notation later in the project. I think that this chapter supplied adequate analysis and that a suitable number of references were used.

5.1.2 Design

Initially in the Design chapter I restated my aims in terms of the software being built. I then proceeded to explain which Development Methodology would be adopted. This was a pivotal moment because it was here that I decided that tried and tested methodologies did not suit this particular project. This revelation meant that the forthcoming work schedule would prove to be somewhat burdensome, although a challenge.

I then turned focus to justifying my choice of hardware and control methods. After which I decided on what programming language would be used and which platform the software would be designed for. Initially it was hoped that it might be multi-platform, and it may be, but this was not tested. Although, the majority of mini-games will definitely be compatible on all platforms, including Linux flavours that support Flash Player.

To conclude the chapter I reported plans for the user interface. I believe this was one of the sections where my work may be perceived to be lackadaisical. I think that this would be an unfair assessment. When writing the section I was aware that I would be trying to describe my design choices in the next chapter; my intentions were to avoid having too much repetition in the report.

5.1.3 Implementation and Testing

In this final chapter I began by summarising how external data would be incorporated into the game and then justified my unorthodox use of CuePoints. I was pleased that I found a way to provide time data, as it was very disheartening when I realised that my early incarnation of the game was simply unable to have this functionality to the level of accuracy that was required for synchronous playback.

The following section discussed the Java software and how it could be bridged with ActionScript. What I had learnt about Sockets in my Web Development unit was very useful at this point and some of the code could be reconfigured to fit the purpose of the server.

The chapter concluded with a section detailing the evolutionary process of the features in the experiments and the main game, finishing with an overview of the Chord Testing game and finally, the mobile game. Until reading the following quote from Harmonix's Greg LoPiccolo, I was shocked to learn that some people trying the games preferred a standard scrolling mechanism to the experimental concepts I had created.

"I think they're great games - but the thing we learned from those games is that an abstract concept is very hard for people to get their heads around," (Androvich, 2008)

Testing was a challenge for me, as I did not really have a final game. I regretted not recording data whilst I was building iterations of my prototypes but I am pleased that I managed to resolve many of the issues that I faced during the development phase. It was difficult to convey my primary concerns into the report, which were the countless limitations imposed during developing the animation. It was because of Adobe AIR's severe performance drop that I decided to rewrite all of the software I had created up to that point to target Flash Player instead. This decision cost me a lot of time and also made me lose many of the file-system capabilities that I had tested in preparation for building the main game. Nevertheless I stand by my judgement, as I know that the game would be unplayable if I had continued to use AIR. In future work I will be more cautious in using brand new technologies.

5.2 Lessons Learned

Halfway through working on the project it became clear that the initial idea for making lots of abstract mini games was flawed, as it was far too big a project to consider. I think that my major undoing was that I didn't spend enough time planning each aspect of the project before development began. The biggest problem I faced was that it became overwhelmingly convoluted. Reigning in all the work that I had done was an ominous and at times, overbearing task. I feel that the hours spent and effort involved affected important areas of the project, such as testing.

Whilst the report itself may be unstructured or too informal in parts, I feel that the amount of research and knowledge gained plus time invested into expressing ideas and developing features can be regarded as strong points. I learnt a huge amount about the subject and myself whilst working on this project. I hope to further my research into the investigation of teaching through playing with particular regard to MIDI Connectivity.

Chapter 6. Conclusion

This chapter provides an overview of the project and concludes with recommendations for future work.

6.1 Overview

The objective of this assignment was to research and develop software for teaching users to play the guitar. This was achieved but did leave a need for further work to create a more comprehensive package. Extensive research was carried out resulting in several prototypes being created, together with Desktop and Mobile game demos. It is expected that the code produced could be merged or reused to help to assemble a fully functional guitar-learning suite.

Some major limitations were encountered during the project are listed below-

- Adobe AIR has very poor framerate capabilities compared to Adobe Flash Player
- Cuepoints created dynamically by ActionScript, or embedded using FLVTool are mostly inaccurate to the degree of at least $1/10^{th}$ of a second
- It is not possible to light frets above the sixth fret position on the neck of the Yamaha EZ-AG
- As of version 10.4.8 onwards, Mac OS X does not support the *com.apple.audio.midi* Java package out of the box. Which means that for applications to access the *CoreMIDI* system, the Mac Java Midi Subsystem must be downloaded and installed from http://www.humatic.de/htools/mmj.htm

6.2 Future Development

This research could be extended by collecting a rewriting the code exclusively in a high level programming language such as C++. This would eliminate many of the timing and animation difficulties that needed to be circumvented and allow better integration into the user's operating system for methods such as file access. Some of the features developed in this project have potential for further research and development, particularly the fret-lighting feature and finger position information.

Now that the principal features have been established, the game would benefit greatly from having more content added into it; a mixture of video tutorials and music videos combined with a story structure and level system would enhance the playability of the game further create motivation for the player.

Utilising the new touch-screen technology found on modern mobile phones could be a very effective learning tool for the user when they are away from their guitar. Creating a virtual mobile fretboard would allow users to practice techniques such as Chord positions in the same way that they would on a real guitar.

Chapter 7. References

Amory A, (2007), *Game Object Model version II: A theoretical framework for educational game development*, Education Tech Research Dev vol. 55 (1) pp. 51-77

Bates M A, (2000), *A Comparison of teaching Guitar Lessons for Beginners using a Multimedia Approach with a Conventional Approach*

Becta, (2006), *Engagement and motivation in games development process*, Available online at: http://becta.org.uk [Accessed on: 13/01/2008]

Bradshaw (2005), *Computer Game 'Playability' – 'Learning' Through Gameplay Design*

Cabral G, Zanforlin I, Lima R, Santana H, (2001) *Playing along with D'Accord Guitar,* Available online at: http://gsd.ime.usp.br/sbcm/2001/papers/rGiordano_Cabral.pdf [Accessed on: 17/01/2008]

Cakmakci O, Berard F, Coutaz J, (2003), *An Augmented Reality Based Learning Assistant for Electric Bass Guitar,* Available online at: http://students.creol.ucf.edu/ozan/arbass.pdf [Accessed on: 03/02/2008]

Clark D, (2006), *Games and E-Learning*, Available online at: http://www.caspianlearning.co.uk/downloads/documents/Whtp_caspian_games_1.1.pdf [Accessed on: 10/02/2008)

Clark D, (2007), *Games, Motivation and Learning*, Available online at: http://www.caspianlearning.co.uk/downloads/documents/Whtp_Games_Motivation_Learning.pdf [Accessed on: 10/02/2008)

Csiksczentmihalyi M, (1988), *Flow: The Psychology of Optimal Experience*

Dambrauskas P, Jantsch EA, (2003), *Real-Time Guitar chord recognition,* Available online at: http://citeseer.ist.psu.edu/628194 [Accessed on: 06/08/2008]

Dannerberg R B, Sanchez M, Joseph A, Capell P, Joseph R, Saul R, (1989), *A Computer-Based Multi-Media Tutor for Beginning Piano Students*

Denis G, Jouvelot P, (2004), *Building the Case for Video Games in Music Education*

Denis G, Jouvelot P, (2006), *Motivation-Driven Educational Game Design: Applying Best Practices to Music Education*

Dobra, (2009), *Harmonix Founder Talks About Real and Virtual Music,* Available online at: http://news.softpedia.com/news/Harmonix-Founder-Talks-About-Real-and-Virtual-Music-101911.shtml, [Accessed on: 27/02/2009]

Draper S, (2000), *Analysing fun as a candidate software requirement*, Available online at: http://www.psy.gla.ac.uk/~steve/fun.html [Accessed on: 14/02/2008]

Dümpelmann, (2003), *Interview With Tommy Victor and Scott Ian*, Available online at: http://www.harkin.org/prong/interview/tommy001.shtml [Accessed on: 27/02/2009]

Fober D, Letz S, Orlarey Y, (2004), *IMUTUS – An Interactive Music Tuition System*

Galarneau, (2005), *Authentic Learning Experiences Through Play: Simulations and the Construction of Knowledge*

Gershenfeld, (1999), *When Things Start to Think*

Gunter G A, Kenny R F, Vick E H, (2006), *A Case for a Formal Design Paradigm for Serious Games*

Gunter G A, Kenny R F, Vick E H, (2007*), Taking educational games seriously; using the RETAIN model to design endogenous fantasy into standalone educational games*

HistoryLink, (2005), *'Hendrix, Jimi (1942-1970)'*, Available online at: http://www.historylink.org/index.cfm?DisplayPage=output.cfm&file_id=2498, [Accessed on: 27/02/2009]

I-MAESTRO, (2006), *Interactive Multimedia Environment for Technology Enhanced Music Education and Creative Collaborative Composition and Performance,* Available online at: http://i-maestro.org [Accessed on: 19/01/2008]

Isaacson E J, (2001), *Music Learning Online: Evaluating the Promise*

Jones M G, (1998), *Creating Electronic Learning Environments: Games, Flow, and the User Interface*

Kaio, (2008), *AS3 FPS Counter*, Available online at: http://kaioa.com/node/83 [Accessed on: 10/01/2008]

Kieras D E, (1997). *Task analysis and the design of functionality*

Laurel B, (1993). *Computers as theater*

Liarokapis F, (2005), *Augmented Reality Scenarios for Guitar Learning*

Manjoo F, (2007), *How "Guitar Hero" saved guitar music,* Available online at: http://machinist.salon.com/feature/2007/08/15/guitar_hero/ [Accessed on: 14/03/2008]

Mochal T., (2001*), Waterfall vs. RAD: How to pick the right method for your study* http://articles.techrepublic.com.com/5100-22-1044102.html?tag=rbxccnbtr1 (accessed 02 15, 2008).

Paras B, Bizzocchi J, (2005), *Game, Motivation, and Effective Learning: An Integrated Model for Educational Game Design*

Percial G, Wang Y, Tzanetakis, (2007), *Effective Use of Multimedia for Computer-Assisted Musical Instrument Tutoring*

Pichlmair M, Kayali F, (2007), *Levels of Sound: On the Principles of Interactivity in Music Video Games,* DiGRA 2007 Conference

Prensky M, (2001), *Digital Game Based Learning*

Reed S, (1995), *A Comparison of Software Development Methodologies*, Available online at: http://www.stsc.hill.af.mil/crosstalk/1995/01/Comparis.asp [Accessed on: 21/02/2008]

Reilly D, (1998), *Java Network Programming FAQ*, Available online at: http://www.davidreilly.com/java/java_network_programming/#1.1 [Accessed on: 19/08/2008]

Rieber L P, (1996), *Seriously considering play: Designing interactive learning environments based on the blending of microworlds, simulations, and games*

Roads C, (1996), *The computer music tutorial*

Royce W, (1995), *Managing the Development of Large Software Systems*

Ruben, Brent D, (1999), *Simulations, Games, and Experience-Based Learning: The Quest for a New Paradigm for Teaching and Learning,* Available online at: http://sag.sagepub.com/cgi/content/abstract/30/4/498 [Accessed on: 02/02/2008]

Salen, Zimmerman, (2003), *Rules of Play: Game Design Fundamentals*

Sheppard M, (2003), *Digital Stringless Guitar*

Sloper, (2003), *Striking The Balance*, Available online at: http://www.sloperama.com/downlode/StrikingTheBalance.ppt [Accessed on: 05/09/2008]

Technion, (1997), *Chapter II: ATM Networking,* Available online at: http://www.comnet.technion.ac.il/projects/winter97/java_atm/Book/2.html [Accessed on: 17/03/2008]

Xin M, Watts C, Sharlin E, (2007), *Let's Get Physical: How Physical Control Methods Make Games Fun*