Dancing Monkeys

(Automated creation of step files for Dance Dance Revolution)

MEng Individual Project Report
18th June 2003

Karl O'Keeffe

Project Supervisor: Dr Iain Phillips
Second Marker: Dr Philippa Gardner

Project home page: http://www.monket.net/dancingmonkeys/

1 Abstract

Dance Dance Revolution is a new style of arcade game controlled entirely by players' feet. They dance in time with music by following steps displayed on a screen. Current incarnations of this game have fixed song selections. Each song has a step track, consisting of a stream of arrows unique to it, which follows the timing and style of that piece of music.

The aim of this project is to extract features from music that allow the creation of new step tracks for arbitrary songs, without human intervention. The result will be a program, which given a piece of music, will automatically generate a fun and playable step track unique to that song. This step track can then be used in a PC based Dance Dance Revolution simulator.

This report details the design of such a program and evaluates its effectiveness. The development of the program has lead to the discovery of new and powerful algorithms in music analysis, as well as successfully demonstrating the power of computers in developing creative works.

2 Acknowledgements

I would like to thank my supervisor, Iain Phillips, for his sharp insight into the problems encountered, and his uncanny ability to immediately suggest a good solution for each one.

I would also like to thank the developers of the Dance Dance Revolution simulators, Dance With Intensity and StepMania, for providing two free and excellent front ends to my project.

For allowing me to use his maximal clique finding code, my immense thanks to Dr Robert Mitchell of Reading University.

Lastly I must thank Pete and Alex for agreeing to be a pair of dancing monkeys in the project demonstration.

3 Table of Contents

4 Introduction

4.1 Motivation for this Project

Dance Dance Revolution (DDR) is a very popular Japanese arcade game, which has a player dance in time to different pieces of music. A player stands on a metal pad with four sensors corresponding to up, down, left and right arrows. The player steps on these sensors as prompted by a scrolling set of arrows on the display. In effect the player dances along to a piece of music. An in-depth description of the Dance Dance Revolution game can be found in the background section of this report.

The success of this new style of arcade has led to many new versions (or mixes as they are known) being created for all world markets. Each mix has a fixed set of songs the player can choose to dance to. To play different songs requires a different version of the arcade machine.

The popularity of this arcade game has led to conversions and simulators being written for all major platforms. Many of the simulators have been created such that they can be extended with new songs.

To add a new song to a simulator requires the difficult and arduous task of constructing a step track by hand. This requires discovering the exact number of Beats Per Minute (BPM) for that piece of music. Beats Per Minute is a measure of the speed of a piece of music, and can be simply explained as the number of times you would tap your foot to a piece of music in one minute. This BPM calculation must be correct to within 0.05 BPM or the arrows begin to shift in relation to the music. The exact timing of beats must also be found, to an accuracy of around 0.05 seconds, in order to ensure the correct positioning of the stream of arrows.

Then a suitable pattern of arrows must be chosen and tested. This is difficult to do and creating a step track that is fun and interesting requires real creativity and talent.

Due to its time-consuming and difficult nature most players avoid this and instead download copies of the steps used in the arcade machines, never taking advantage of the extensibility of the simulators.

This project aims to rectify this by providing a completely automated process for creating step tracks to any song. In order to achieve this, the inherent difficulties in step track creation must be overcome.

These difficulties include: Automatically computing the BPM of a piece of music to within the required tolerance, correctly locating the position of the beats, and creativity and talent when generating the patterns of arrows.

The main motivation for this project can be found in the behaviours of players recently introduced to Dance Dance Revolution. Many people, on first encountering the game, immediately inquire about the possibility of dancing to

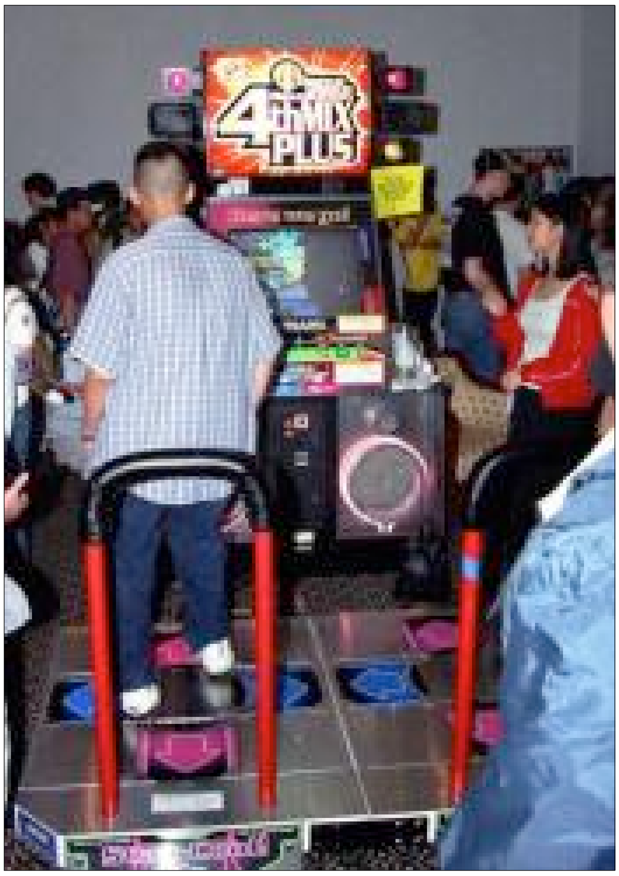


Figure 1 – Photograph of a Dance Dance Revolution arcade machine. The two floor pads can be clearly seen. Unfortunately the screen in the centre is too small to make out the moving arrows.

their favourite tune. This is invariably not possible, due to the small selection of songs provided by the arcade machines. This project aims to overcome this, by providing a simple mechanism whereby any song can be added to DDR.

4.2 Areas of Major Interest

The first two major problems, finding the BPM and the position of the beats, could be considered solved problems. There are many algorithms in the field of music analysis that can extract the BPM and beat positions from a piece of music, but there are none that can do so to the accuracy required by this project. In order to reach the required accuracy new ideas and algorithms will be required.

The task of producing fun and interesting step tracks on the other hand, is an entirely unique problem. The need to imbue the computer with an element of creative talent vastly increases the difficulty of this task. The step tracks produced by this project will be required to compete with those developed and refined over a period of time by professionals.

In order to produce competent step tracks, the computer will need to understand the structure and events present in the music. This will involve automated extraction of musical structure, a difficult task with very little research coverage.

Other musical features such as the structure of beats, and the position of held notes, would also aid the creation of step tracks.

4.3 Deeper Objectives

There are three objectives to this project that are deeper than just filling a gap in one gaming genre.

This project aims to show that academic and state of the art music analysis techniques can be applied to real world problems in an efficient and reliable manner. Part of this will be to show that disparate areas of research can be combined together successfully.

Separate from the previous aim is the desire to show the feasibility of computers generating creative works, in the form of producing new, original, and interesting step tracks without user assistance.

Finally the project aims to be more than just a research study of feasibility. The result of successful completion will be an application of sufficient reliability and quality that it can be released to, and used by, untrained computer users. This report only lightly touches on this facet of the project, as creating usable polished applications is a reasonably well solved problem, and the least interesting area of this project.

4.4 Report Structure

The rest of this report will show that this project fulfilled its objectives, and in doing so developed some new, and very successful, music analysis techniques. It is structured as follows:

Background
This section gives an in depth explanation of the Dance Dance Revolution game and the structure of the step tracks used in it. This is followed by a discussion of the main literature on beat extraction and BPM calculation, selected literature on feature extraction from music, and finally a brief discussion of similar projects.

Design and Implementation
This section gives a brief overview of the project design and architecture, followed by in depth discussions of the interesting or problematic areas of project implementation. Trivial and/or uninteresting areas of the project are not mentioned and can be considered to have been implemented successfully.

Testing
This section states the tests performed and the results given in a series of tests to choose the best parameters for various algorithms. Followed by justification of the applicability of the Bron-Kerbosch clique finding algorithm and the best algorithm for representing a beat.

Evaluation
In this section, a quantitative and qualitative evaluation of the final product is made. This includes testing the effectiveness of the music analysis algorithms, along with a critical evaluation of the quality of the step tracks generated.

Conclusion
This final section covers the key aspects of the project and details possible future work.

5 Background

This section first explains the Dance Dance Revolution game mechanics, along with the structure of its step tracks. This is followed by a discussion of the most applicable literature in music analysis, for both finding the BPM and extracting musical features. Lastly this section covers projects of a similar nature.

5.1 Dance Dance Revolution

5.1.1 Explanation of the Game Mechanics

To really understand this project first requires an understanding of how the Dance Dance Revolution (DDR) game functions. It is a very simple game; a player stands on a pad with four arrows: up, down, left and right. The screen in front of them has four stationary arrows across the top. A stream of arrows rises up from the bottom of the screen in time with the music. When a moving arrow overlaps a stationary arrow the player must step on the corresponding arrow on the pad. In effect they dance to the music.

The arrows are placed such that they are hit in time with the music. Most of the arrows occur directly on a beat in the song, but on higher difficulty level arrows can occur between beats. These arrows are coloured differently to help players distinguish them. As well as stepping on individual arrows at a time there are jumps, where two arrows reach the top simultaneously and the player must jump in order to land on both of them at the same time. Newer versions, or mixes, of DDR contain freeze arrows, so called because the player must keep their foot on the pad until the end of the elongated freeze arrow has reached the top of the screen.

Each mix of the DDR game comes with a limited selection of songs. Each song can be played in one of three difficulty modes: Basic, Trick or Maniac, with each

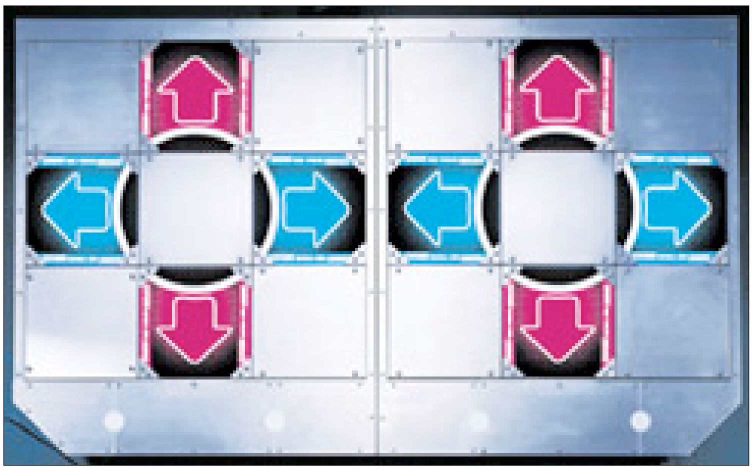


Figure 2 – The floor pads used in the Dance Dance Revolution arcade machines. Only the areas with arrows are buttons. Pads similar to these can be purchased for home use.

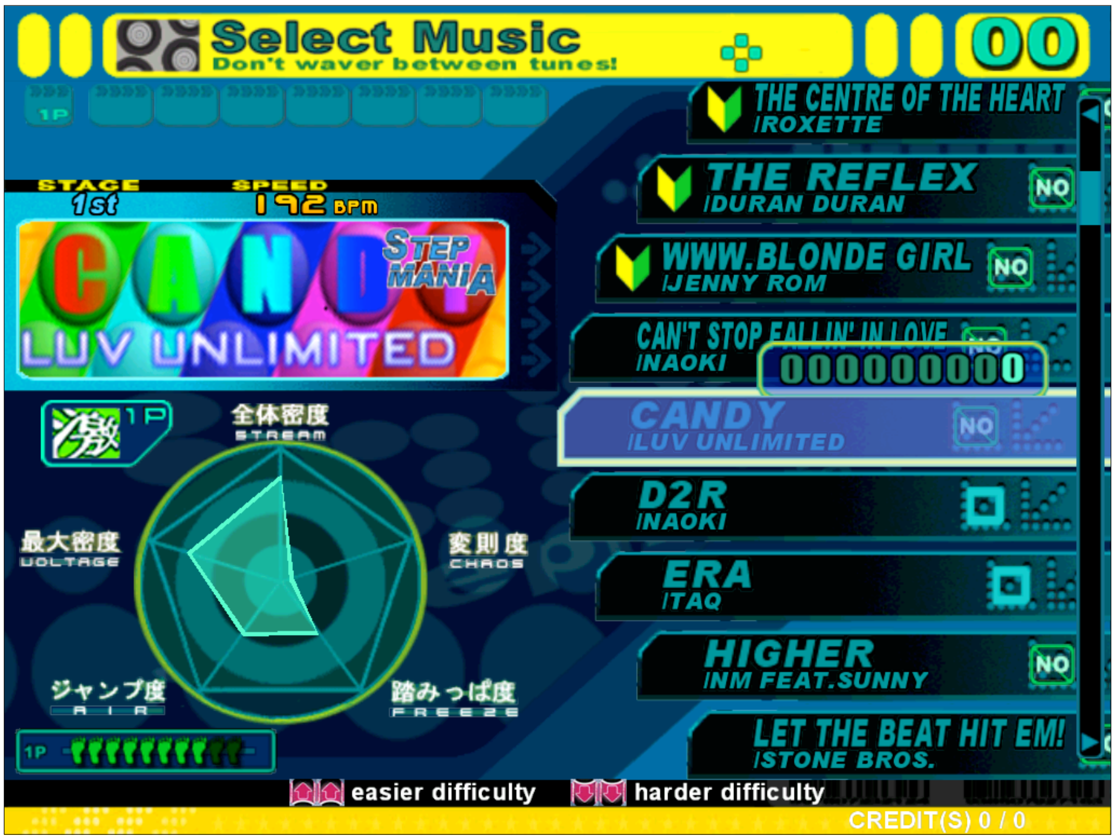


Figure 3 - A screenshot showing the groove radar and foot rating when choosing a song. The groove radar can be seen as the pentagon in the lower left. The foot rating for the song is below and too the left of this.

mode giving a different patterns of arrows. For every song, each of these modes is also given a difficulty rating in feet, ranging from 1 to 9. A one foot song can be played even by complete novices, whereas nine foot songs can only be bested after months of play.

The latest mixes of Dance Dance Revolution contain a revised difficulty rating system. The difficulty modes have now been renamed to Light, Standard and Heavy. While the foot rating system has been replaced with a Groove Radar. This is a graphical representation of the difficulty and composition of the songs. It measures five attributes: Voltage, Stream, Chaos, Freeze, and Air. Voltage measures the peak density of steps in a given song, while Stream measures the overall density of the steps. Chaos shows the irregularity of steps and Freeze displays the number of freeze arrows. Air gives a measure of the number of jumps in a song.

Dance Dance Revolution arcade machines have two pads side by side allowing two players to dance at the same time. They also allow a single player to utilise both pads by selecting doubles mode. In this mode there are eight arrows at the top of the screen rather than the usual four, and the player must dance on both pads in order to successfully hit every arrow.

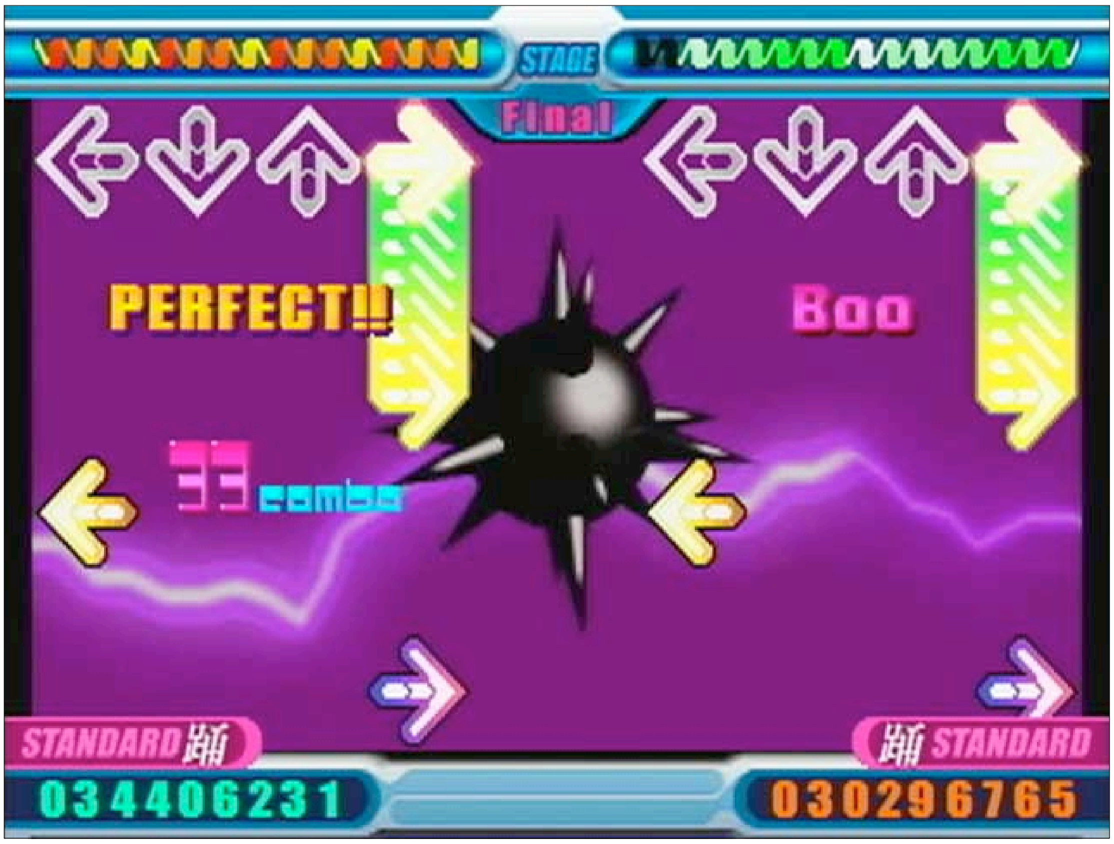


Figure 4 - A screenshot of a two player Dance Dance Revolution game showing freeze arrows and offbeats. The freeze arrows are visible as the stretched arrows near the top of the screen.

The style of music provided by the DDR arcade machines is either cheesy remixes of well known songs, or specially written dance music. The defining factor of all DDR songs is that they have a very prominent and stable beat. A few of the songs change their beat timings, but this is uncommon.

Dance Dance Revolution is the type of game that can only really be understood by playing it; a written description doesn't do it justice. More information on DDR can be found at the DDR Freak website [1] .

5.1.2 DDR Simulators for the PC

Due to its popularity, Dance Dance Revolution has been cloned many times, and versions of it exist on all major computing platforms. These clones, or simulators, are not limited to a fixed selection of songs. Any piece of music can be used as a dance track, so long as the DDR simulator is given a text file, called a step file, specifying the beat of the music and the patterns of arrows to be displayed.

For the Windows operating system the best DDR simulators are Dance With Intensity (DWI) [2] and StepMania (SM) [3] . It is possible to purchase dance mat

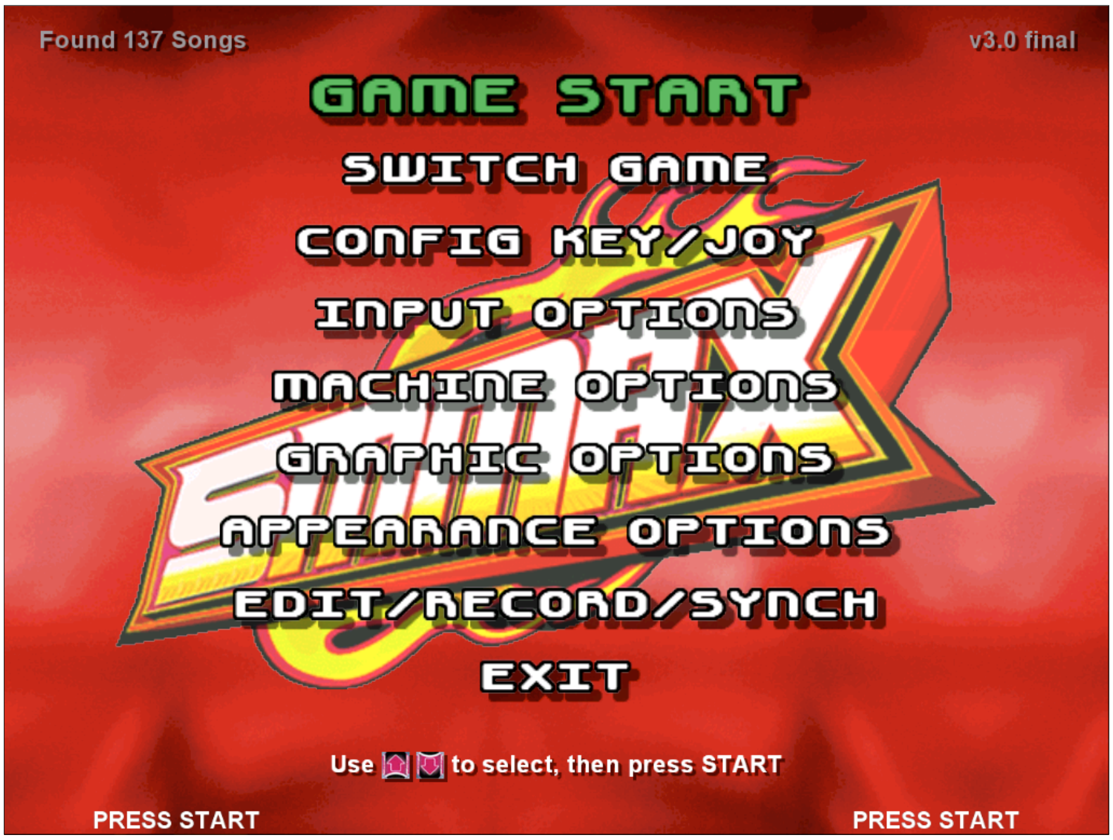


Figure 5 – A screenshot of the loading screen for the StepManis DDR simulator on the PC.

controllers for the home, which can be used with both Dance With Intensity and StepMania to further emulate the arcade experience.

Both DWI and SM accept a variety of formats for the music and step files. Both accept music encoded as .wav, .mp3 or .ogg files. More information on the Ogg Vorbis [4] and Mp3 [5] encodings can be found in the references section.

Dance With Intensity only accepts step files in its own .dwi file format. Whereas StepMania is more accepting, it will import arrow data from .dwi, .sm, .bms and .ksf files. Detailed information on the .dwi [6] , .sm [7] and .bms [8] file formats is available in the references section.

Application	Music files formats	Step file formats
Dance With Intensity	.wav, .mp3, .ogg	.dwi
StepMania	.wav, .mp3, .ogg	.dwi, .sm, .bms, .ksf

Table 1 - Accepted music and step file formats for DWI and SM.

The formats of the various step files, although different, require the same basic pieces of information. These are:

- Beats per Minute (BPM) of the music track. Beats per Minute is a measure of the speed of the music. If you foot tapped along to a piece of

```
#TITLE:DXY!;
#ARTIST:TAQ;
#BPM:148;
#GAP:100;

#SINGLE:BASIC:4:000000000000000040000000600000000B0000000B0000000…
#SINGLE:ANOTHER:6:00000000000000004020202060202020B0208020B08020…
#SINGLE:MANIAC:8:000000000000000040206080402060800(42)0(42)0(68)…
#DOUBLE:BASIC:4:000000000000000040804080206020608000602080006020…
#DOUBLE:ANOTHER:6:0000000000000000000000006060606080600000006080…
#DOUBLE:MANIAC:8:00000000000000004020806080206000000000006086482…
```

Figure 6 – Excerpt from a .dwi step file. The last six lines are the arrow patterns for various difficulty levels.

music, the number of times you tap your foot every minute is the BPM of that piece of music. The music used in DDR can have a BPM rating of anywhere from 90 BPM to 300 BPM, with the average being somewhere around 150 BPM.

- Gap value, specifying the amount of time from the beginning of the music to the first beat. This value allows Dance With Intensity or StepMania to synchronise the arrows so that they reach the top of the screen in time with each beat.
- The pattern of arrows to dance to. For certain step file formats more than one stream of arrows can be placed in a file, each corresponding to a different difficulty level.

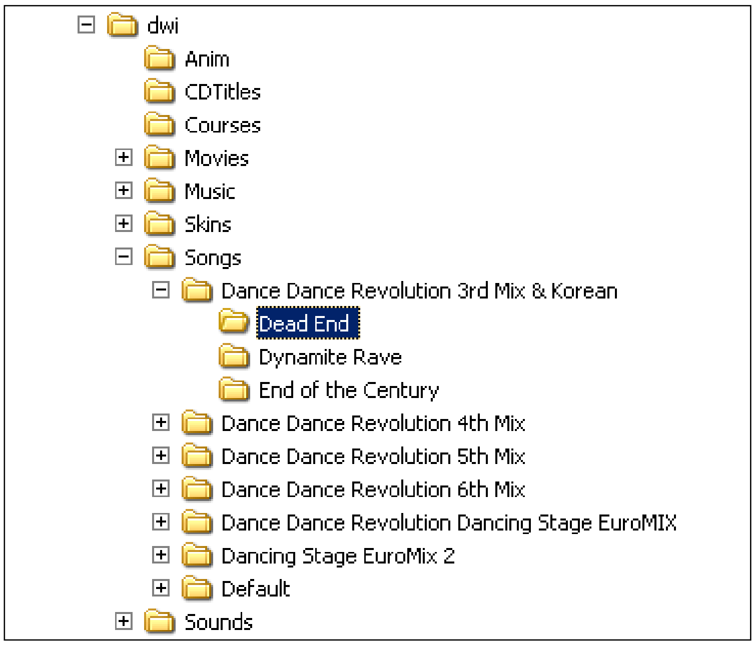
The files also accept a lot of other extraneous information, such as the song name, title, artist, or background effects to use while dancing.

Both DWI and SM use the same directory structure for storing the music, step files and other associated files such as background images. This structure consists of a Songs folder as a subdirectory of the main program. The songs folder contains a number of sub directories corresponding to groups of music, such as a group for music from the original DDR mix and a group for music from the European mix. Within each group there is a folder for each song. Each song folder contains the music, the step file and background or banner images.

5.1.3 Rules of DDR and its Step Tracks

The style of music used in DDR always has a very prominent and simple beat, which is usually exaggerated compared to normal music. This beat must be accurate to ensure the arrow patterns line up correctly. Human drumming is not accurate enough, thus the propensity for electronic music and remixes in DDR. The average song only last about two minutes, which is shorter than normal music due to the physical exertion required.

Most songs in DDR are structured similarly. Three or so base arrow patterns, or motifs, are repeated, with slight variations throughout the song. Each of these motifs corresponds to a different section of the song.



```
⊟ dwi
    Anim
    CDTitles
    Courses
  ⊞ Movies
  ⊞ Music
  ⊞ Skins
  ⊟ Songs
    ⊟ Dance Dance Revolution 3rd Mix & Korean
        Dead End
        Dynamite Rave
        End of the Century
    ⊞ Dance Dance Revolution 4th Mix
    ⊞ Dance Dance Revolution 5th Mix
    ⊞ Dance Dance Revolution 6th Mix
    ⊞ Dance Dance Revolution Dancing Stage EuroMIX
    ⊞ Dancing Stage EuroMix 2
    ⊞ Default
  ⊞ Sounds
```

Figure 7 - Diagram of the directory structure used for storing step files and music for DWI and SM. Each of the deepest nested folders contains the music, step file and background for a single DDR track.

The variation in these patterns is usually manifested in a mirroring or rotating of the arrows. For example if a pattern consisted of alternating left/right arrows, it could be changed to alternating up/down arrows.

Freeze arrows are usually placed where notes are held for a period of time, making the player keep their feet steady in time with the music. In many cases while a single freeze arrow is held, other arrows still scroll up the screen. In this case the player must continue to hit the other steps with their free foot.

The arrow pattern for each difficulty level of a single song, generally builds upon the previous one. It is easy to spot the same overall structuring of arrow patterns, even though the individual motifs may have changed.

In DDR only some patterns of arrows are valid. For instance it is not possible to have more than two arrows reach the top of the screen at the same time. Valid arrows are either single arrows or jumps. Jumps can consist of any combination of two arrows. Simple jumps are the up-down and left-right pairs. Complex jumps, or corner jumps, are the combinations of one vertical arrow with one horizontal arrow.

Arrows normally occur exactly on a beat, or halfway between two beats. Although in some more challenging step tracks, arrows do occur at other, more arbitrary, positions.

Finally, the arrow patterns of official songs have a 'flow' to them, which means it is usually possible for a player to follow the patterns with alternating feet and never be forced to cross their legs or awkwardly shift their weight.

5.2 Beat Detection

One of the most important parts of this project is the algorithm used to extract the beats, and thus BPM, from the music. Without an accurate estimate of the BPM the arrows from the step file will not occur on time with the beats in the music and it will be impossible to dance to the step track. Beat detection is a Digital Signal Processing (DSP) problem. A brief introduction to Digital Signal Processing is available in the appendix and explains many of the terms used to those who are not familiar with them. More information on Digital Signal Processing is also available at the DSP Guru website [9] .

There are many approaches to detecting the beats and calculating the BPM of a piece of music. The main literature in this area is discussed below. An overview of each paper and the algorithm used within it is given below, along with brief discussion of it accuracy and applicability. The reader is encouraged to read each of the papers in full for more detail, the aim here is give a brief introduction to the many ways in which beat detection can be performed. The papers are listed roughly in order of their perceived importance to this project. All diagrams from within this section come from their respective papers.

5.2.1 Tempo and Beat Analysis of Acoustic Musical Signals

Tempo and Beat Analysis of Acoustic Musical Signals – Eric D. Scheirer [10]

Scheirer's paper is one of the most frequently referenced papers on beat detection from real world (polyphonic) music. The paper goes into detail of the implementation of a fast, close to real time, beat detection system for music of any genre. It also provides experimental results of its accuracy.

The algorithm works by first dividing the music into six different frequency bands using a filterbank. This filterbank can be constructed using a number of low-pass, band-pass and high-pass filters.

The envelope of each frequency band is then calculated. The envelope is a highly smoothed representation of the positive values in a waveform.

The differentials of each of the six envelopes are calculated, they are highest where the slopes in the envelope are steepest. The peaks of the differentials would give a good estimate of the beats in the music, but the algorithm in the paper uses a different method.

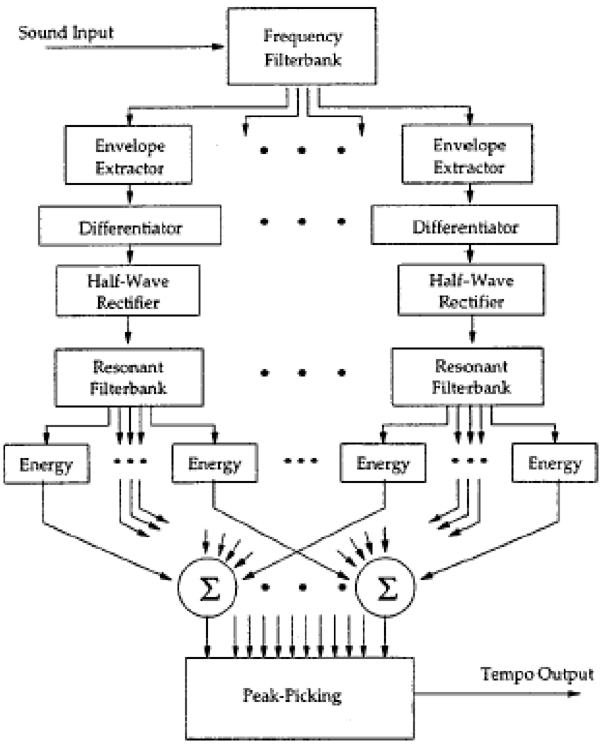


Figure 8 - Schematic view of the very popular beat detection algorithm proposed by Scheirer.

Each differential is passed to a bank of comb filter resonators. In each bank of resonators, one of the comb filters will phase lock with the signal, where the resonant frequency of the filter matches the periodic modulation of the differential.

The outputs of the all of the comb filters are examined to see which ones have phase locked, and this information is tabulated for each frequency band. Summing this data across the frequency bands gives a tempo (BPM) estimate for the music. Referring back to the peak points in the comb filters allows the exact occurrence of each beat to be determined.

The beat detection strategy used in this paper has demonstrated high accuracy and has been implemented many times by different parties. It can cope with a wide variety of music genres and fits the requirements of this project. The speed of the algorithm may also be advantageous to this project.

The algorithm is very complex and may be time consuming to implement. By working with music in a stream, it fails to take advantage of the ability to analyse all of the music as one element. This means that while the accuracy may be good enough to tap along with users in real time; it may not be able to determine the BPM to a sufficient accuracy for this project.

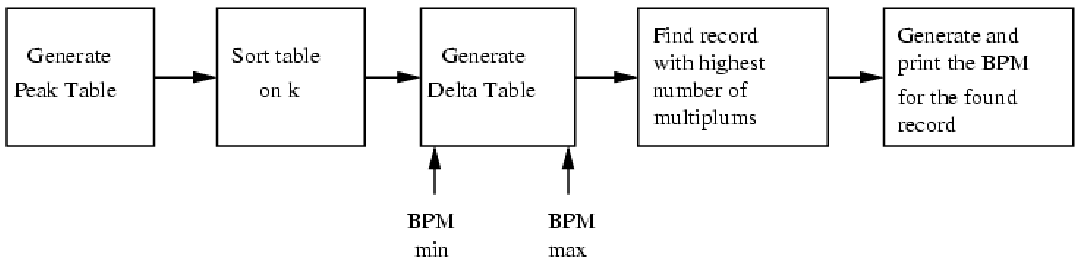

Generate Peak Table
Sort table on k
Generate Delta Table
Find record with highest number of multiplums
Generate and print the BPM for the found record
BPM min
BPM max

Figure 9 - Schematic view of the simple beat detection algorithm proposed by Arentz.

5.2.2 Beat Extraction from Digital Music

Beat Extraction from Digital Music – Will Archer Arentz [11]

This paper is a useful reference as it takes the same perspective as this project; it is concerned with calculating the BPM of a song in a non-real time manner. In doing so is able to provide a high level of accuracy.

The algorithm proposed in this paper has two phases. Initially the music is filtered to remove 95% of the samples with the lowest amplitude. This is a very rough way of extracting the peaks that is unaffected by the quiet- or loud-ness of the music. The remaining samples can then be treated as beats in the music.

The next step is to determine which BPM matches most closely to the remaining samples. This is done by testing all possible BPMs within a certain range, 60-160 BPM in this paper, and counting the number of beats that occur exactly on time with that BPM. The calculation of which beats occur for a given BPM is done by specifying the BPM in terms of the number of samples between each beat, say k. Every beat is then tested to see the number of other beats that occur at a time which is an exact multiple of k above or below the time of the beat being tested. For each value of k the corresponding number of matching beats is stored in a table.

Once all possible values of k within the BPM range have been calculated, the entry with the highest number of corresponding beats is taken to be the correct value for the number of samples between each beat. It is then trivial to convert this to a BPM value.

The approach used by this paper is very simplistic, devoid of any theory or application of research into the human perception of music. Despite this, it generates accurate results, to within 0.1 BPM for 95% of the music tested.

The paper makes no mention of calculating the positions of the beats, and is thus of no help in determining the gap value of a piece of music. The lack of theoretical basic to the algorithm is also disturbing, but is lessened by the extremely good empirical results.

5.2.3 The Beat Spectrum: A New Approach to Rhythm Analysis

The Beat Spectrum: A New Approach to Rhythm Analysis – Foote, Uchihashi [12]

This paper provides yet another approach to determining the beats, and therefore the BPM of a piece of music. The method discussed in this paper uses self-similarity to characterise the rhythm of the music, giving a beat-spectrum from which the BPM can be extracted.

The initial step in the algorithm is to compute some measure of self-similarity. The method used in the paper is based on the distance-matrix, in the paper Visualising Music and Audio using Self Similarity [14] , which is discussed in the next section on feature extraction. The music waveform is divided into overlapping frames a few hundred samples wide. A fast Fourier transform is performed on each frame. The logarithmic of the magnitude of each result gives the power spectrum for each frame.

A 2-dimensional representation of the similarity of the music is created by plotting the similarity of each power spectrum with respect to every other. This gives a plot of time against time, where the value at every point represents the similarity of the music at the two given times.

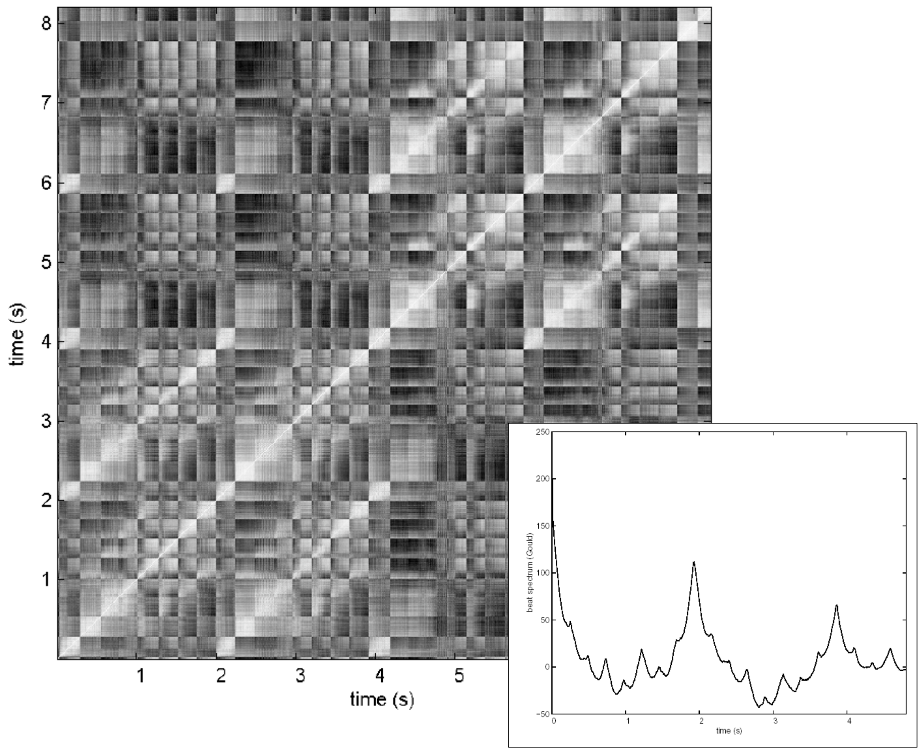


Figure 10 - Self-Similarity matrix for Bach's Prelude No.1 with its associated beat spectrum. The diagonal white lines in the matrix correspond to the peaks in the beat spectrum. The peak at two seconds in the beat spectrum shows that each bar of music lasts two seconds.

The beat spectrum itself is determined by summing all points in the similarity matrix that have the same difference in time between each axis. This is equivalent to summing along each diagonal line in the self-similarity matrix.

The peaks in the beat spectrum can be used to calculate the BPM of the music. Taking the distance between zero and the highest peak in the spectrum and dividing this by the number of peaks from zero to the highest peak, gives you the interval between each peak; from this it is trivial to calculate the BPM.

There are many advantages to the approach taken by this paper. By solely using self-similarity and auto-correlation, the algorithm can work with all genres of music equally well. The paper also suggests that the algorithm gives very robust results and is not prone to mistakes or failure. Changes in BPM during a piece of music are detected easily.

Although initially this algorithm looks very promising in its applicability for this project, there are a couple of areas where it seems to be lacking. The first is with the accuracy with which the BPM can be calculated. The paper claims accuracy to within 1%, unfortunately this is not nearly accurate enough for this project. The algorithm also seems computationally very expensive; the paper makes no mention of the efficiency of the algorithm in this respect.

5.3 Extracting Features from Music

There is little appropriate literature on the subject of feature extraction from music that can be applied to this project. Most algorithms used to perform feature extraction are used in information retrieval systems, where the idea is to reduce a piece of music to a small vector of relevant data. Unfortunately this does not help in this project, where the aim is to analyse the music to discover the times at which various features occur. The few available papers with relevant ideas are covered below.

5.3.1 Locating Singing Voice Segments within Music Signals

Locating Singing Voice Segments within Music Signals – Berenzweig, Ellis [13]

This paper presents a technique for locating portions of music during which vocals are present. The results may be useful for applying different arrow patterns to vocal or instrumental sections.

The algorithm proposed in the paper uses a complex system of speech recognisers to detect individual vocal sounds. Each speech recogniser is tuned to respond highly to a certain phonetic sound, with sharp cut-offs on either side of that sound. To music, the speech recognisers respond with very low levels over a longer time period. This information can be used to classify whether a section of music contains vocals.

A simple Hidden Markov Model (a type of state machine, see elsewhere for a full definition [17]) is then used to interpret the data from the speech recognisers and label sections of data as either vocal or instrumental.

The algorithm used in this paper achieved 80% accuracy in determining vocal or instrumental sections. As the accuracy of determining these sections is not critical to this project, this algorithm could be used.

The main draw back with this algorithm, at least with respect to this project, is its complexity. It requires a number of speech recognisers which are able to detect various phonetic sounds, this is computationally expensive and a much simpler method of doing things is likely to be just as useful.

5.3.2 Visualising Music and Audio using Self-Similarity

Visualising Music and Audio using Self-Similarity – Jonathan Foote [14]

The Beat Spectrum: A New Approach to Rhythm Analysis [12] explains the use of self-similarity for BPM extraction. Self-similarity is also a very useful tool for extracting structure from musical pieces. This paper provides a more detailed overview of the uses of self-similarity.

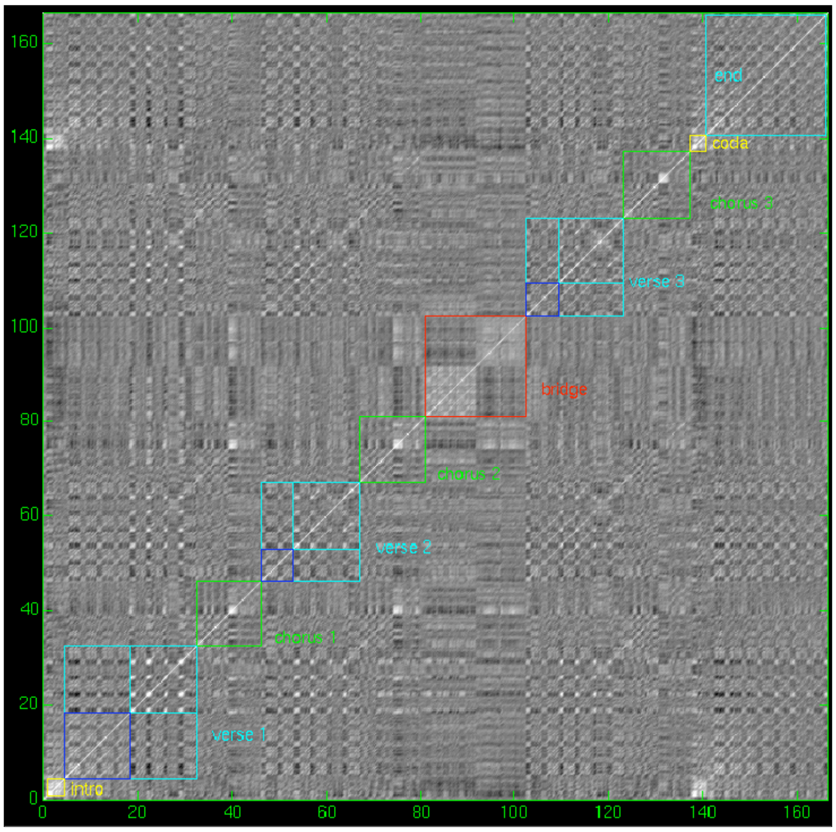


Figure 11 – Self-similarity matrix of Day Tripper by the Beatles, annotated with structural information. Self similarity can be used to detect structure such as repetition for use in arrow generation.

The basics of computing audio self-similarity were given in the discussion of The Beat Spectrum. This paper provides a much more in-depth discussion of exactly how it is calculated and the reader is encouraged to read it.

The interesting point in this paper is the discussion of the use of audio self-similarity for discovering features in the music. Many examples are discussed in which it is shown that self-similarity shows the potential to be able to automatically detect repetition and structure in a piece of music. It may be possible to automatically determine the locations in time of each chorus in the music. This is extremely useful for this project, as it provides a time structure to link the structure of the arrow pattern to. Unfortunately the paper only theorises about the possibility of extracting the structure, no implementation ideas are discussed.

The paper also shows the potential of self-similarity for discovering parts of music which lend themselves to be used for freeze arrow placement. Areas that do not change are coloured bright white on the similarity plots. These unchanging sections of music are perfect for placing freeze arrows.

Self-similarity provides a very useful method of analysing the structure of music, and it is particularly applicable to this project. Its main flaw is its computational cost, working out the similarity of every section of music to every other section is a time consuming process.

5.3.3 Riddim: A Rhythm Analysis and Decomposition Tool

Riddim: A Rhythm Analysis and Decomposition Tool Based on Independent Subspace Analysis – Iroro Fred Onome Orife [15]

This paper is very long and in-depth look at computer analysis of music and rhythm in general. It provides a detailed description of an algorithm based on that proposed by Eric D. Scheirer, but with greater applicability to music with a more complex rhythmic structure.

It is of interest mainly because of a technique known as Independent Subspace Analysis. This is a technique for dividing a complex stream of music into its constituent parts. It will divide a piece containing many different instruments into individual streams comprised of the sound generated by each instrument. Once the streams have been separated they can be analysed independently.

Although this paper is very detailed in its background and algorithmic explanation, it lacks much information about the quality of the results obtained. The main aim of this paper also differs somewhat from the aims of this project. Nevertheless there is valuable insight to be gained from the paper, and the use of Independent Subspace Analysis may be useful for feature extraction.

5.3.4 An Automated Structural Analysis of Music

Listening to "Naima": An Automated Structural Analysis of Music from Recorded Audio - Roger B. Dannenberg [16]

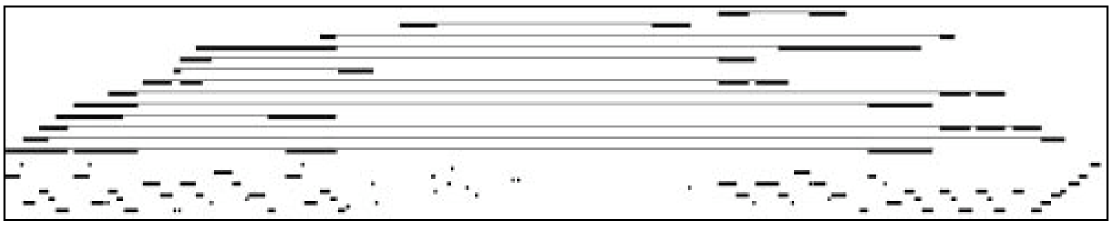


Figure 12 - A visualisation of the clusters found by the Automated Structural Analysis of music paper. The thick black sections joined by thin lines indicate similar clusters. The small black dots at the bottom indicate extracted pitches.

This paper describes an approach to automatically deriving the structure of musical pieces. This is of importance to this project, as the identification of repetition in music will allow the correct repetition of motifs in the step track.

The approach of the algorithm is to extract the melody from a piece of music, before transcribing this into a series of pitches and their associated durations.

Then a self-similarity matrix, like that described in Visualising Music and Audio using Self-Similarity [14] , is constructed. From this, areas of similarity (diagonal bright lines) are extracted into clusters of similar areas.

These clusters of similar areas show which points in time each point in the music is similar to. This representation is very complex, and in order to simplify it a naïve parsing algorithm is used to extract a one-dimensional sectioning of the music. This places each point in time as a member of a single section.

The algorithm proposed in this paper has only been tried on three songs, and only produced successful results for two of these. It also currently requires human assistance and tweaking to provide reliable results. Because of this, in its current form, it is not directly applicable to this project, but many of its ideas could be useful.

5.4 Arrow Pattern Generation

It is not really surprising that there is no current literature on the problem of automatically generating Dance Dance Revolution arrows given an arbitrary piece of music. The only mention of arrow generation is two projects with a similar theme to this one.

5.4.1 Project Proposal – Dance Dance Revolution

An engineering project with scope somewhat overlapping this project's has been proposed at MIT [18] . It is mainly concerned with implementing a complete system, from music, to dance pad, to video game. As the system is to be implemented almost entirely in hardware, the logic for constructing the patterns of arrows has to be relatively simple. The approach suggested in the proposal is to keep a number of different patterns in memory and randomly pick a stream of

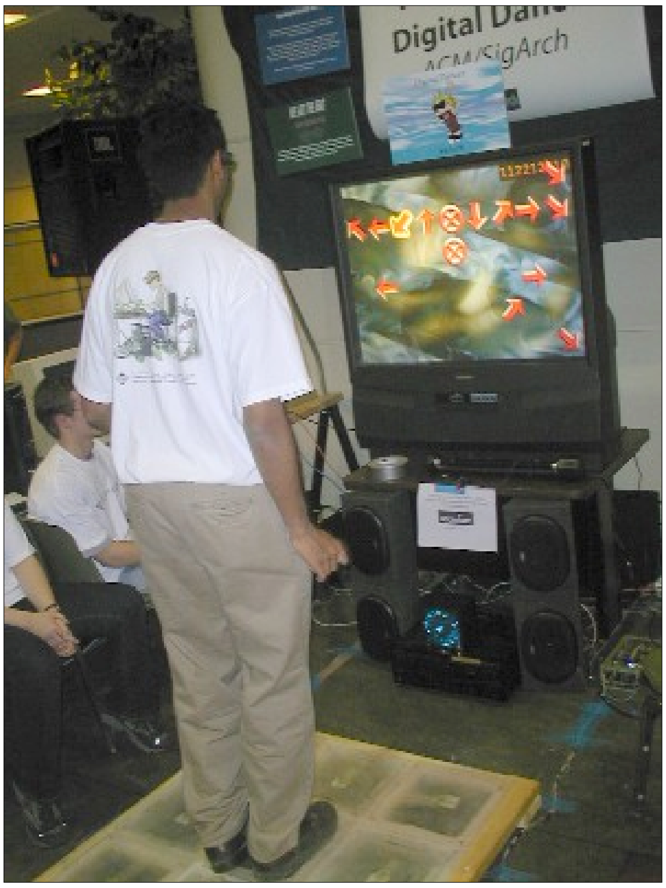


Figure 13 – The Digital Dancer project from the University of Illinois takes a similar, but more hardware based, approach to the same problem this project addresses.

these patterns. The system makes no analysis of the music to aid its arrow generation.

Unfortunately there is no information available as to the results of the completed project, its efficiency or accuracy.

5.4.2 Digital Dancer

This is project by the ACM group at the University of Illinois [19] , attempting a similar hardware based approach to the MIT project proposal. So far they have created the dance mats and a program to display the streaming arrows, but have not yet completed the code to create the stream in real time.

They use a simple beat detection scheme similar to the first half of algorithm proposed in Tempo and Beat Analysis by Eric Scheirer, simplified due to constraints of the hardware they are using.

Unfortunately there is no information present on their web site to explain how they are approaching the task of generating the arrow patterns.

6 Design and Implementation

This section of the report gives a very brief description of the design and architecture of the project, before detailing the most difficult and interesting problems faced and the approaches taken to solving them. This is not a complete account of all areas of the project, many small details are omitted and can be assumed to have successfully implemented. This approach was taken in order to increase the readability of this report.

6.1 Design and System Architecture

The system is divided into five main areas, as shown in the figure below.

The first section decodes music files from a given selection of formats; in the final implementation mp3s and wave files are supported. The output of this section is processed by the second section in order to extract the BPM and gap value for that piece of music. With knowledge of the position of the beats, the music is processed by the third section to extract relevant features, including the structure of the song in repeating sections. Using the extracted features, the fourth section generates the pattern of arrows for each difficulty level. Finally the fifth section outputs these arrows into text files in either the .dwi or .sm format.

6.2 Implementation Overview

The project was implemented in MATLAB [23] , a matrix based programming language with excellent support for digital signal processing. To produce the final application, the MATLAB code is first converted to C, and then compiled for the Windows platform. This can be done automatically by MATLAB.

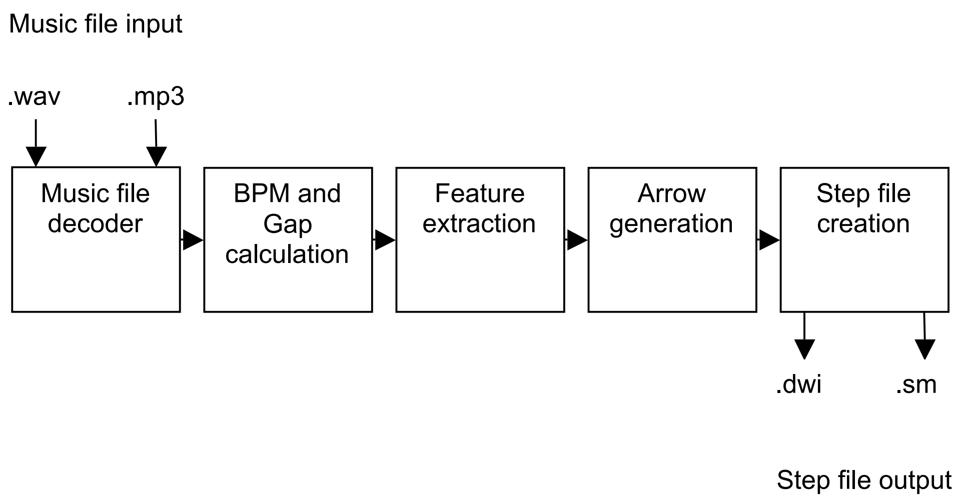


Figure 14 - Overall system architecture for this project.

The music decoding is accomplished through a call to LAME [24] , a command line mp3 encoding and decoding application. The decoded mp3s are then natively loaded into MATLAB.

6.3 Calculating Beats per Minute and Gap Value

Once loaded into MATLAB, the song is processed to extract the BPM and gap value. It is assumed up front that any song has same BPM all the way through and that the BPM is very accurate (i.e. machine generated). This assumption turns out to hold for most pop music and styles of music used in DDR. The assumption does heavily limit the possible input range, but allows for the use of faster, more accurate algorithms.

The technique used in this project is similar to that in the Beat Extraction from Digital Music paper [11] , but was developed independently. It is assumed that beats are always signalled by the playing of a bass instrument. This is a valid assumption in this project, due to the type of music used in DDR.

Bass instruments are low frequency but have a high energy. This means they have the greatest amplitude, and thus correspond to the peaks, in the waveform. Therefore picking the highest peaks from the waveform will give the time at which each beat occurs. By measuring the gaps between these peaks it is possible to work out the length in time of each beat, and thus the BPM.

It turns out not to be so simple in practice. When a drum is hit (or any instrument is played) it does not just create a single peak in the waveform. It creates a number of very rapid peaks, depending on its frequency, these peaks start very high and decay rapidly. Therefore simple peak picking would give many peaks for each beat. One way to avoid this happening is to create a smoothed version of the waveform, where all the rapid oscillations are removed, but the overall shape of the waveform is preserved.

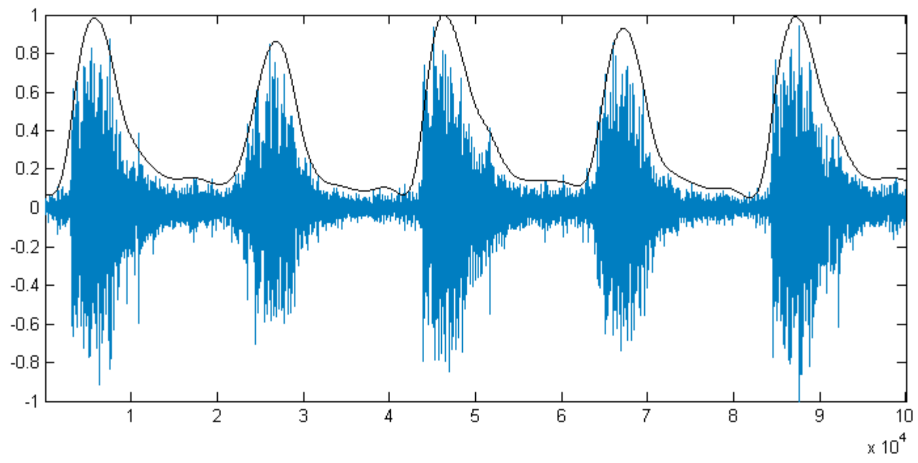


Figure 15 – This images shows a waveform, and what can be considered a smoothed representation of that waveform (the dark line). Smoothing allows more reliable peak picking.

In this project, this smoothing is performed by filtering the waveform with a low pass filter. This is a standard signal processing technique for removing unwanted high frequencies. A simple explanation of a digital filter is given in the appendix. The testing section covers how the most appropriate filter was chosen.

This filtering adds another complication; it shifts the position of elements in the waveform. Different frequencies in the waveform are shifted by different amounts. Therefore it is no longer possible to know where the peaks in the smoothed waveform actually were in the original waveform. This is not a problem for BPM detection, as peaks from the same instrument will all be shifted by the same amount, so the relative distances between beats stays the same.

But the shifting means it is no longer possible to accurately determine the gap value. There exists a simple but infrequently used solution: reversing the filtered result and passing it back through the same filter. No more frequencies are removed, but exactly the same amount of shift is applied to each frequency in the opposite direction, thus returning all points to their original positions.

This technique is not frequently used because it requires the whole waveform to be known in advance. This is usually not the case in most signal processing applications, but in this project the whole waveform is known so this technique can be applied.

To improve peak picking for songs with both loud and quiet sections, the highest peaks are taken within small sections of music, rather then just the tallest peaks in the whole waveform. This ensures that beats in quiet sections of music can still contribute to finding the BPM.

Even with a single peak per beat, the simple method of calculating the interval between beats (and thus BPM) by taking the mean (or median) of the distances between peaks gives an estimate of the BPM with an error of around ±3 BPM. The variability in this estimate is too great for use in DDR. The error really needs to be less than ±0.05 BPM.

Therefore an alternate strategy is taken: For every possible BPM, within reasonable limits, it is computed how well the beat data fits that BPM. Then the BPM value which fits 'best' is chosen.

This requires some measure of the fitness of a particular BPM value. For every BPM, it is simple to calculate the time interval between beats that it represents. Therefore for a single BPM, and thus interval, a gap value can be computed for each beat. This is simply the remainder of each beat position after dividing by the interval. In a plot of gap values against time, a dense horizontal collection of points indicates that there are many beats that agree on the same gap value, and thus many that support the BPM being tested. The BPM with the most support is taken to be the correct BPM.

The difficulty with this method is in reliably detecting dense and horizontal collections of points in a manner that allows a single measure of fitness to be

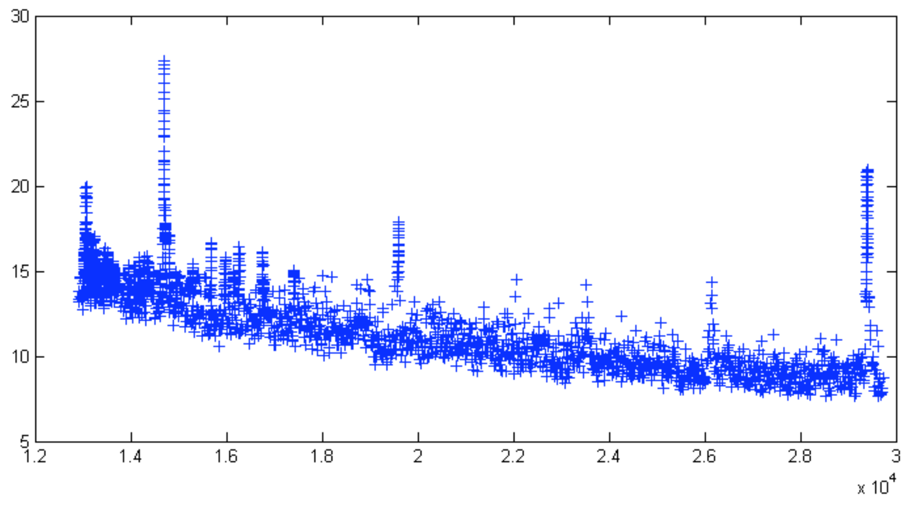


Figure 16 – This plot shows the fitness of each BPM against the interval between beats for that BPM. The bias towards higher BPMs (on the left side of the plot) can clearly be seen as a rising of the minimum fitness found.

computed. The approach taken in the final implementation of the project is described below, and as a side effect it also calculates the optimum gap value.

A window is moved over the range of possible gap value, and at every point the number of calculated gap values within the window is counted. These counts give a measure of the fitness of each possible gap value. A bell shaped window is used, which ensures denser collections of computed gap values are considered fitter. Additionally biasing counts based on the strength of the beats producing the gap values reduces the effects of spurious beats.

The maximum fitness of the gap values is used as the fitness of the BPM being tested. Stored with the fitness of the BPM is the value of the fittest gap value. Once it has been worked out how 'fit' each BPM is, the fittest is picked as the BPM to use in the step file. If no BPM stands out as being appreciably fitter than any others, than it is concluded that the algorithm has failed to find the correct BPM (if a correct BPM exists). Once the BPM has been determined, the gap value used is the fittest of the gap values for that BPM.

It turns out that the fitness function used in the implementation of this project has a bias towards higher BPMs. This occurs because high BPMs have smaller intervals between beats. Therefore there is less possible distance between the gap values we compute, and thus there will be more gap values in each window.

To rectify this it is possible to fit a polynomial curve to the results of the fitness function, and use this to normalise the fitness function to a flat line. This ensures any peaks on the fitness function can now be compared using their absolute value.

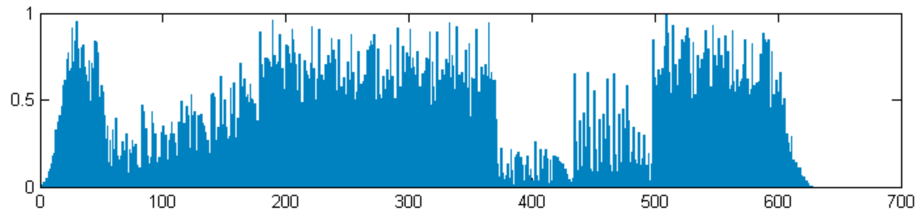


Figure 17 – The above bar graph shows the varying energy levels of each beat and offbeat throughout a test song.

6.4 Feature Extraction

6.4.1 Representation of a Beat

In order to simplify the arrow creation section of this project, each beat (and offbeat) is transformed from its waveform into a much simpler canonical representation. This representation can then be used to aid the arrow placement algorithm.

One very simple piece of information can extract for each beat of music is its energy. This is given by converting the waveform to a positive representation through squaring, and then calculating the area underneath this new squared waveform.

Low frequency instruments, such as drums and bass guitars, have higher energy signals than other instruments. This fact combined with the squaring of the waveform means that low frequency (bass) instruments dominate the energy representation.

This does not cause too much of a problem in the context of this project, as the events that are most interesting are beat orientated, which tend to be events generated by bass instruments.

These are not the only events of interest though, and in the energy representation of a beat, other events such as singing or melodies are drowned out by bass instruments.

Another method of creating a feature representation of each beat is a voice analysis technique known as Mel Frequency Cepstral Coefficients (MFCC) [25] . This technique works out, for a period of time (such as one beat), a feature vector of thirteen elements. These elements correspond to events in different bands of frequencies.

This richer representation allows for information, such as the presence of events in the frequency band in which singing occurs, to be used when placing arrows.

An MFCC library routine was tested in this project for its applicability in generating a representation of each beat. Unfortunately it was found that this

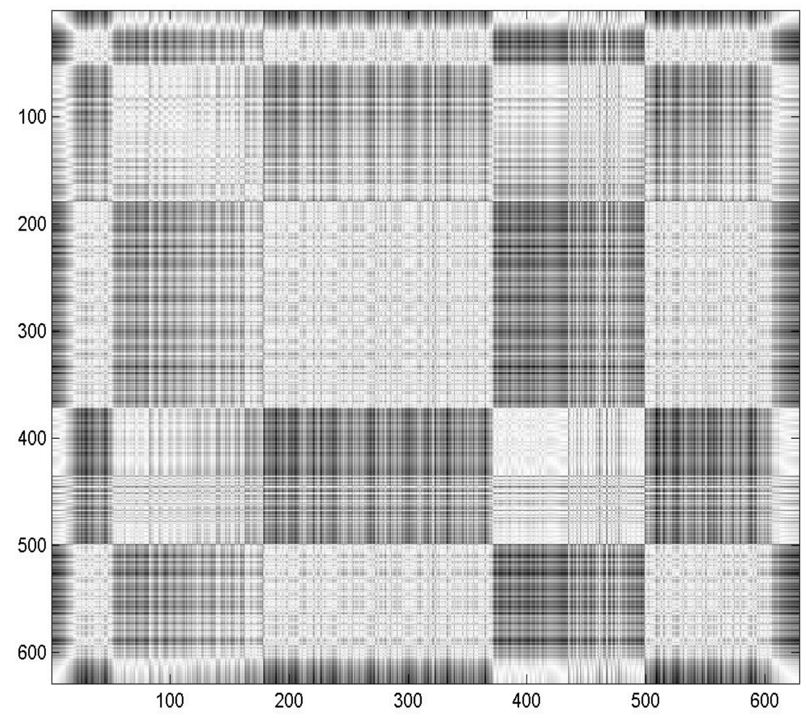


Figure 18 - Self Similarity matrix for All the Things She Said by Tatu. The top left indicates the start of the song. The light grey squares along the centre diagonal indicate different sections of the song.

representation produced less accurate results in later functions (such as computing self-similarity and grouping beats) than the simple measure of energy. Further discussion of this can be found in the testing section.

6.4.2 Extracting the Structure of a Song

An important feature in official DDR step files is structure. Most official tracks feature a particular pattern of four to eight arrows that are repeated in the main areas of the song. Small sections of music that sound the same usually have similar arrow patterns.

The step files generated by this program should exhibit a similar structure. This can be accomplished by working out which areas of a given song are similar to other areas.

The paper Visualising Music and Audio using Self-Similarity [14] provides a method of producing images, showing which instants of a piece of music are similar to other instants of that music, known as self-similarity matrices. This seems highly applicable to problem faced here. An explanation of the algorithm used is given in the background section.

The instant of time for computing similarity in this project is a half a beat. This gives a high enough resolution to make informed decisions about placing offbeat

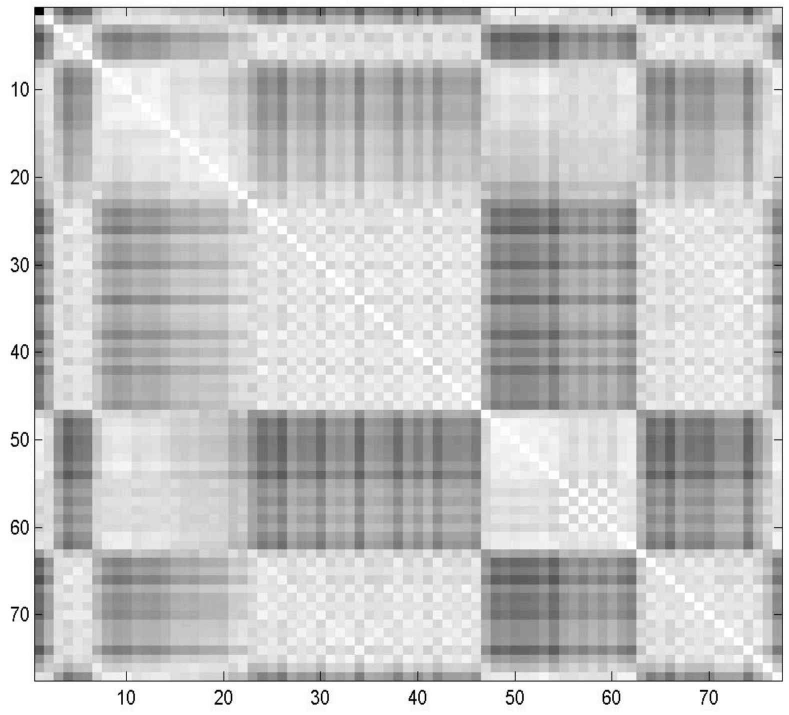


Figure 19 – A similarity plot of All the Things She Said by Tatu, with each pixel being one bar of music. This representation can be converted to a linear structural representation of the song.

arrows. Using the proposed algorithm we can compute which half beats are similar to other half beats. Although rather than use MFCC or a Fourier transform as our feature vector for each half beat, we use energy as discussed above. Along with producing clearer images of self-similarity using energy as our representation makes the algorithm a lot faster. The different feature vectors are discussed further in the testing section.

It turns out that just knowing the similarity between half beats is not altogether too useful. It is on too fine a scale to make easy decisions about the patterns of arrows. Half beat time intervals are too short to be able to compute sensible grouping for a song.

Most music is divided into fixed size bars. Major changes in the music usually occur only on the boundaries between these bars. Thus if we take our time period to be the bars of the music we can produce a more useful self-similarity matrix. This introduces the sub problem of dividing a piece of music into bars.

For this project, the assumption is made that all music is in 4/4 time. This means that there are always four beats (and thus eight half beats) in every bar. This may seem restrictive, but it turns out to be a reasonably valid assumption. Almost all of the music used in DDR is in 4/4 time.

Therefore the problem is now simply finding when the bars of music start. With only four beats in every bar, there are only four possibilities to check. Each can be tried and the best, with respect to some measure, can be taken as the correct one.

This requires some method of measuring how good a bar division each position gives. The algorithm chosen in this project is, for each position, to calculate a self-similarity matrix with a time period of a whole bar. The self-similarity matrix with sharpest differences between bars is likely to be correct division of music into bars.

There are a few different approaches to quantifying this measure of sharpness. The testing section details the approaches tried, and the final method settled on.

Knowing the division of the song in bars, and a measure of how similar each bar is to every other bar gives a very detailed account of the structure of the music. Unfortunately this representation is too complex to use for placing arrows. A simpler, linear, representation is needed, where each bar belongs to a single group, and every member of that group is similar to every other.

For these groups it is unimportant if members of a group are contiguous bars or not. In order to produce the best translation of the self-similarity matrix to these groups, the final representation should contain as many large groups of bars as possible. In order to decide if a bar is similar enough to another bar, simple thresholding can be performed on the self-similarity matrix.

If we treat each beat as a node on a graph, and draw an arc between each pair of nodes if the nodes are similar, we have an undirected graph representation of our data.

This problem of finding the division into as many large groups as possible is then equivalent to solving the minimum clique partition problem of the graph representation. A clique is sub section of nodes in which every node in the sub section is connected to every other node in the sub section. A minimum clique partition is a division of a graph into cliques such that the minimum number of divisions is made.

The minimum clique partition is a well-known NP complete problem, and a very computationally intensive one at that. No efficient implementation algorithms have been proposed, even for small graphs (There is one paper that claims to have a polynomial time algorithm, but this would imply P = NP and thus is not likely to work in practice).

The naïve, or brute force, approach to this problem, where all possibilities are tried and the best taken works for very small graphs of around eight or less nodes. After this the algorithm takes much too long to execute. The number of nodes (or bars) in the songs in this project range from fifty to one hundred nodes, so the naïve algorithm cannot be used.

Therefore the project instead uses an efficient but non optimal algorithm. The testing section gives a qualitative evaluation of how un-optimal the solution really is.

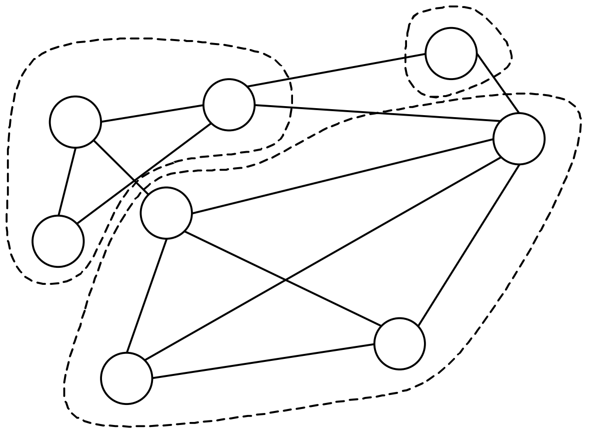


Figure 20 - A minimal clique partition of an acyclic graph. The problem of computing the linear structure of a song can be considered equivalent to finding a minimal clique partition.

Part of the minimum clique partition problem is the problem of finding all of the maximal cliques in the graph. A maximal clique is a clique that cannot be made any bigger by adding new nodes while remaining a clique.

Finding all maximal cliques in a graph is also a well-known NP complete problem.

But efficient algorithms are available for reasonably sized graphs, with fifty nodes. The testing section shows that, even though graphs with up to one hundred nodes are found in this project, the efficient algorithms are still applicable.

The algorithm used in this project is that proposed by Bron and Kerbosch [22] . A highly efficient implementation of the Bron Kerbosch algorithm has been developed by Dr Richard Mitchell (a lecturer in cybernetics at Reading) in MATLAB, and it is this implementation that is used in this project.

Once the maximal cliques have been found, a greedy algorithm is used to divide the song into groups. This is non-optimal but efficient. It works by taking the biggest clique as a group; from all remaining cliques, removing nodes already in a group; and repeating this until only single element or empty cliques are left.

A refinement is made to ensure that songs which are similar all the way through are still given interesting structure. If the biggest clique covers a large percentage of the song, it is ignored and smaller cliques which will give better partitioning are chosen.

The greedy approach works well in practice, and reliably divides songs in a way that human listeners would agree with. This is discussed in greater detail in the testing section. The lack of a guaranteed optimal solution does not seem to be a problem.

The use of this grouping for arrow creation is discussed later.

6.4.3 Pauses

In DDR arrow streams can pause for a period of time and stop scrolling up the screen. This technique is used when there is a sudden stop in the music and a period of little or no noise. In order to recreate this, a technique for locating these silent periods in each song is required.

These silent periods can be found by looking for contiguous beats that have an energy representation below a certain threshold. In keeping with the arrow generation procedure, which is explained later, pauses must start and end on the boundaries between bars.

This assumption that pauses only ever occur for whole bars at a time is reasonably valid. It does not hold in all cases, but in many songs it holds true. Making this simplifying assumption helps reduce erroneous positioning of pauses in many songs.

The assumption is made because music is normally structured such that major events, like a pause, only occur for whole bears at a time. Of course not all music follows this rule.

6.4.4 Freezes

Later DDR mixes introduced a new type of arrow, called a freeze arrow. A freeze arrow is simply an elongated version of a standard arrow, which is long enough to be spread over many beats. The player must keep their foot on the pad for the whole length of time the freeze arrow is scrolling past the top of the screen. They are mainly used when a note is held for a period of time.

To enable their use in this project mechanism is required for detecting when a note is being held. This can be formulated alternately as the periods of time in which no new musical events are heard. I.e. any sound that is still perceivable must have come from the last note played.

This formulation is not exactly equivalent to finding notes that have been held. For the purposes of this project it will suffice, as accurately determining when notes have been started and stopped in polyphonic music is an unsolved problem.

Freeze arrows are treated more like normal arrows than pauses, so there is no restriction on them occurring only for the length of full bars. There is a requirement that they start and end exactly on a full beat, rather than starting on an offbeat.

The algorithm used for finding periods of time with no music events, and thus periods of time that can be used for freeze arrows, is given below.

A musical event is designated as an onset occurring in the waveform. This ensures that major events such as a new note being played are detected, but slow building of volume is not counted as an event. This is in keeping with the use of freeze arrows in official DDR songs.

To locate uneventful periods, and thus freezes, we wish to find all onsets, no matter what the instrument used. Therefore it is not possible to just check our list of previously found beats, as this only contains bass events.

Onsets are detectable as sharp increases in amplitude in the waveform except, due to high frequency instruments and noise, sharp increases in amplitude occur at all points in the waveform. The smoothed version of the waveform removes these high frequency components and leaves sharp inclines only when onsets occur.

Calculating the gradient at every point in the smoothed waveform will give a new waveform in which the peaks correspond to onsets or musical events. Now finding a possible freeze position is as easy as finding a period of time for which the gradient waveform is below some threshold.

In DDR freeze arrows normally start on the onset of the note which is being held. Therefore the start of the freeze arrows is one beat earlier than the low threshold period.

The use of these freeze positions for placing arrows is explained in the next section.

6.5 Arrow Generation

Once the pertinent features of the music have been extracted, the next stage is for the computer to generate arrow patterns for each song.

There is no set algorithm to do this, no right or wrong arrows to produce, only a user subjective ‘good’ or ‘bad’ opinion on the generated arrows. Therefore the generation technique used in this project is a collection of rules and heuristics to guide the program and improve its chances of generating a ‘good’ stream of arrows.

This is not the only strategy that could have been taken. Initially it was hoped that the project could train a neural network to produce the arrows. This option was discarded early on for two reasons. It became clear that the music analysis section would require a large portion of time to implement, and that there would be insufficient time to design, train and evaluate a neural network system. Secondly, state of the art for neural networks is not very advanced. The music analysis data would constitute a very large number of inputs and require a very large neural network before any non-trivial behaviour emerged.

It was recognised that better results could be produced faster by use of hard-coded ad-hoc rules and heuristics. The most interesting of these rules and heuristics are explained below.

6.5.1 Tracking Foot Placement and Ensuring ‘Flow’

After playing DDR for a while, players begin to notice that there is a way to follow the arrows such that your feet seem to flow easily from one arrow to the next. You alternate feet for every arrow and the next one always seems to be in a

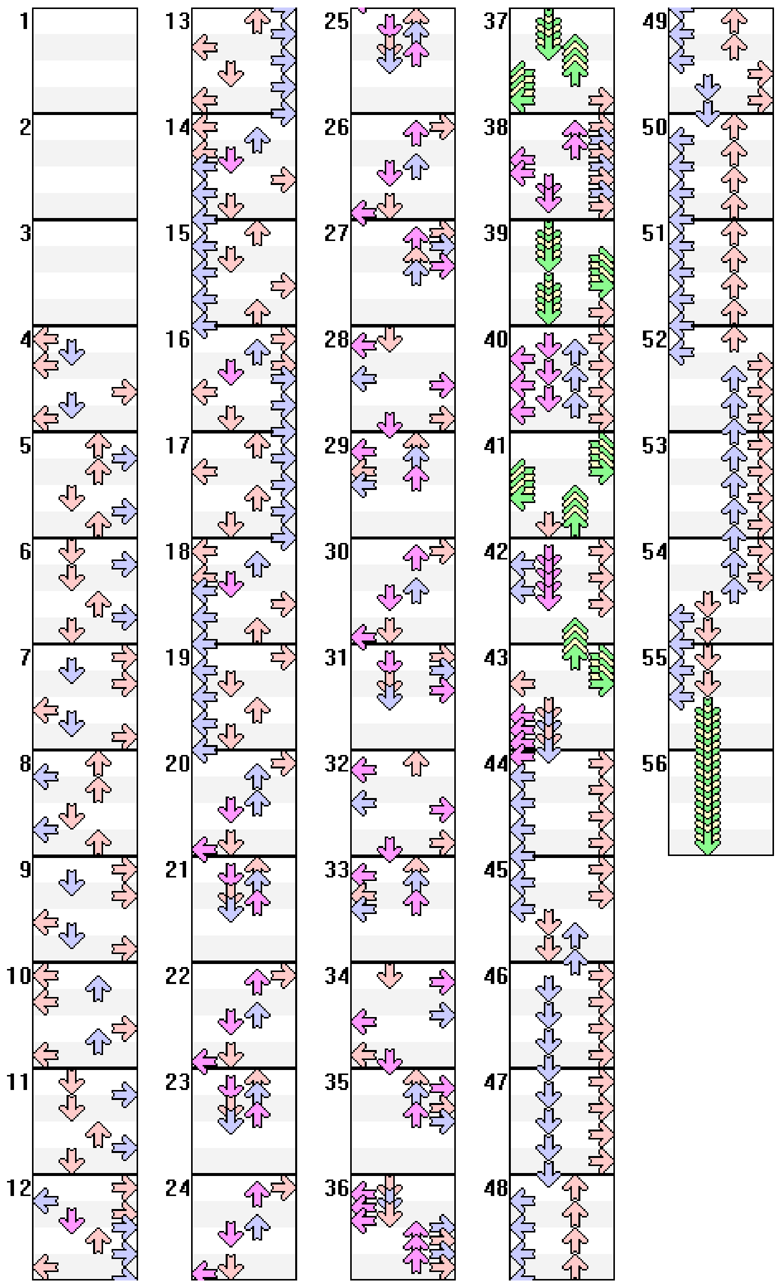


Figure 21 - An example step track from the Rage with Intensity website. The use of repeating but subtly altered patterns can be seen throughout.

natural position. The player never has to awkwardly shift their weight or break their rhythm to follow the pattern.

The initial proposed system for choosing arrow patterns in a manner that would ensure they flow, was to have a database of short arrow patterns and choose a combination of these patterns to build up a complete step track. It was hoped that copying the short arrow patterns directly from the arcade machine, would ensure a high quality of step track.

Unfortunately a study of the patterns used in the arcade machines shows that almost every conceivable pattern of arrow occurs in at least one step track.

Therefore the database would become prohibitively large, and would not provide much guarantee of quality.

As an alternative approach to ensuring flow, a set of rules has been created which captures the knowledge required to allow the program to create flowing arrows. It works by both keeping track of the position of the players feet on the pad, and by a list of rules showing the valid arrow possibilities from each position which will still ensure flow.

With 4 arrows on the floor pad, and 2 feet, there are theoretically 12 possible different positions the player can be in (There are four possible positions to place the first foot, and then only three possible positions to place the second foot, as one arrow is already taken. Thus 4□3 = 12 possible positions).

This is of course assuming that the player only ever stands on the arrows, and avoids all the blank sections of the pad. This assumption turns out not to be true, beginner players almost always stand with both feet in the centre of the pad and only gingerly reach out to step on the arrows. This turns out not to be a problem as beginner step files have large gaps between arrows such that there is always time for a player to shift their weight, and flow is not a problem.

One of the 12 possible positions turns out not to occur in practice. This is the position where the player has their back completely to the screen, with their left foot on the right arrow, and their right foot on the left arrow. Therefore the rules only need to deal with 11 positions.

There are 4 other positions, where the user is almost facing away from the screen, which occur very infrequently and only on higher difficulty levels. These are characterised by the player having either their left foot on the right pad, or their right foot on the left pad.

For each of the 11 positions there are 3 theoretical moves for each foot (stay where it is or move to one of the two unoccupied arrows). In practice, some of these moves never occur, and others occur only infrequently. This knowledge of which moves are possible from which positions is encoded into the project in the form of the probability of moving from one foot position to another,

These probabilities were picked by hand, through prior knowledge of DDR. They could have been better estimated through automatic analysis of existing step files, but due to time constraints this was not possible. It was felt that the increase in accuracy gained would be negligible compared to the effort expended.

6.5.2 Placing Arrows

To decide when is a good time to place an arrow and when a gap should be left in the step file, the canonical representation of a beat is very useful. In this case the canonical representation of a beat is a measure of its overall energy.

A simplistic algorithm, where an arrow is placed if the energy is greater than some threshold, does not scale well to producing arrow patterns at a given difficulty level. It also tends to leave quiet sections completely blank, and populate loud sections with too many arrows.

To overcome this, a different strategy was used. Each difficulty level in DDR can be considered to have a standard range for the possible number of arrows per song. This fact can be used to ensure the total number of arrows placed is correct for a given difficulty level. To do this, first the total energy of each bar is calculated, giving a measure of the loudness of all the bars. This energy rating is then normalised such that the sum of the energy for all bars is equal to the average number of arrows for that difficulty level. Thus this new rating for each bar gives the number of arrows that should be placed for that bar.

To ensure that quiet sections of music get some arrows, and loud sections of music are not overwhelmed with arrows, the ratings are adjusted. For runs of bars with no arrows, it is ensured that an arrow is placed at least every third bar. For runs of bars with eight arrows, and thus no gaps between arrows, it is ensured that there is a gap in the arrows at least every third bar.

6.5.3 Difficulty Levels

The arrow placement algorithm above does a good job of ensuring that the step tracks produced are of the correct difficulty. By assigning the number of arrows globally, it is insured that arrows are well spaced on easy difficulty levels, and clumping of arrows, which would make a step track more difficult than it should be, is avoided.

This is not the whole picture with respect to difficulty of step tracks. There are unwritten rules that state the types and complexity of arrows that can be used for each foot rating. These include avoiding corner jumps, and ensuring that no other arrows are placed while freeze arrows are being held, for easy difficulty songs. Other rules govern the number of off beat arrows that can be placed for each foot rating.

These extra rules are taken care of by conditions on the use of heuristics and rules. These conditions are relatively trivial in the implementation and thus are not detailed here.

6.5.4 Giving a Step File Structure

In DDR step files have a structure. Repeating patterns can be noticed as individual parts of a song repeat. These can be noticed sub consciously while dancing to the step track. Recognition of these repeating patterns gives an air of quality to the step track.

The music analysis grouping of the song into similar bars allows this type of structure to be added to the auto generated step files. For each group the average of all its energy levels is taken, and the average number of arrows per bar in the group is calculated. Using this information, and the rules of flow given above, a bar length pattern of arrows is generated for that group.

This pattern is then applied to all bars in the group. When a consecutive run of bars in the same group is found, variation is added to the pattern to avoid very boring step tracks being generated. This variation consists solely of swapping either the up/down arrows in the pattern, or the left/right arrows in the pattern, or both. This simple technique for variation ensures that the patterns are still immediately recognisable, while altering them enough to be interesting.

To further improve the quality of the step track generated, arrows patterns for groups which contain a large number of bars must contain both up/down and left/right arrows. This ensures that the patterns the player will spend the majority of time dancing to will always be interesting.

6.5.5 Pauses and Freeze Arrows

Placing pauses in the arrow stream is very simple. As pauses can occur at every difficulty level, it is simply a case of annotating the step track with the location and times of each pause. There is no further deliberation to decide which pauses should be included or not, all pauses found are added to the step track. This is a very naïve strategy but works well in practice. The placement of pauses is unaffected by the grouping of the song, a pause in one element of a group does not necessarily mean a pause in all elements of the group.

Placing freeze arrows is a more complex problem. Many official songs in DDR have freeze arrows as part of their repeating patterns. Unfortunately, due to time constraints only a simple freeze arrow placement scheme was implemented. The freeze arrows are simply overlaid onto the existing arrow pattern for the song, and on easier difficulty levels any arrows present at the same time as a freeze are removed. This simple strategy works, but fails to take into account repeating patterns and the flow of arrows.

6.6 Outputting Arrows to a Step File

Once the arrow generation is completed for each of the three difficulty levels (light, standard, and heavy), the pattern of arrows must be output to a file readable by either Dance With Intensity or StepMania.

This involves encoding the patterns in various ways depending on the file format used by the simulator. This encoding is not covered here as it is accomplished by trivial text processing methods.

The step file produced also contains the BPM and gap value, along with the name of the song being processed. The step file and song are then copied to either the Dance With Intensity or StepMania song directory. This signals the end of the processing, and the song can now be danced to using either DDR simulator.

7 Testing

This section covers tests performed during construction of the various algorithms used in this project. These include tests to determine the optimum value of parameters, along with tests of the applicability of the algorithms.

7.1 Parameter Evaluation

This section covers the tests performed to choose the best values for parameters of the music analysis algorithms.

7.1.1 Lowpass Filter Type

When smoothing the waveform for peak picking, there are many different types of low pass filters you can use. To find the most suitable one, three of the best filters were tried with a range of songs. The results are shown below.

		Eminem - Lose Yourself	Foo Figheters - Monkey Wrench	Daft Punk - Aerodynamic	Linkin Park - One Step Closer	The Prodigy - Voodoo People
Butterworth	BPM	171.45079	174.14769	122.86975	190.16818	148.99488
	Gap In Seconds	0.23605	0.11059	0.39644	0.04769	0.23857
	Time Taken	40.141	61.984	50.282	52.25	40.485
Elliptic	BPM	171.45079	174.15915	189.63664	190.16818	148.99488
	Gap In Seconds	0.23599	0.28726	0.30469	0.0478	0.24063
	Time Taken	44.625	64.578	62.078	54.844	44.735
Chebyshev	BPM	171.45079	174.14769	176.76531	142.62613	148.99488
	Gap In Seconds	0.23678	0.1111	0.18943	0.36288	0.23873
	Time Taken	39.578	59.344	63.484	53.657	38.735

For three of the five songs (Lose Yourself, Monkey Wrench, and Voodoo People) the choice of filter made no difference. For Aerodynamic only the Butterworth filter gave the correct BPM, and for One Step Closer the Chebyshev filter gave the incorrect BPM.

Therefore it was decided to use the Butterworth filter in the final implementation as it seems to provide better results. This may be because Butterworth filters generally have a much sharper cut-off and thus less high frequency noise is present in the smoothed waveform.

The data set of only five songs does not definitively show that a Butterworth filter is the best, but it has consistently performed well in day to day usage of the project.

7.1.2 Lowpass Frequency Threshold

The smoothness of the waveform generated by the lowpass filter is based on the frequency cut-off, specified in Hz. A range of different cut-off values were tested with five different songs.

		The Prodigy - Firestarter	Holly Valance - Kiss Kiss	Sho-t feat Brenda - Groove 2001	The Faint - Agenda Suicide	Spiller - Groovjet
1Hz	BPM	141.46707	194.03095	122.534	71.9941	130.011
	Gap In Seconds	0.01259	0.15018	0.13222	0.00045	0.264
	Time Taken	32.297	23.891	22.516	40.437	38.281
5Hz	BPM	70.74488	193.98827	175	72.2872	130.0054
	Gap In Seconds	0.43791	0.29537	0.2698	0.25735	0.2641
	Time Taken	33.937	27.781	27.297	43.609	42.953
10Hz	BPM	141.48976	194.00259	140	144.5586	130.0054
	Gap In Seconds	0.01428	0.14476	0.1018	0.25957	0.2625
	Time Taken	38.86	28.672	30.5	50.171	48.375
15Hz	BPM	141.48976	145.48054	140	144.59016	130.0054
	Gap In Seconds	0.01476	0.08601	0.10172	0.26959	0.26336
	Time Taken	44.5	33.171	31	60.031	53.016
20Hz	BPM	106.1159	96.99769	140	144.5349	130.005
	Gap In Seconds	0.439	0.29617	0.1035	0.25637	0.2644
	Time Taken	56.75	39.11	35.875	72.094	61.359
40Hz	BPM	141.48219	145.49654	140.007	72.2015	130.005
	Gap In Seconds	0.01317	0.08782	21.7329	0.25308	0.2656
	Time Taken	104.985	82.25	76.031	199.735	100.579

The only cut-off that consistently provided the correct BPM for all songs was the 10Hz cut-off. Thus, this is the cut-off used in the final implementation.

7.1.3 Peak Threshold

Once the waveform has been smoothed, the peaks are extracted. In order to reduce the amount of processing, only peaks above a certain amplitude are extracted.

In order to determine this amplitude threshold, five songs were tested with varying thresholds.

		Tatu - All the Things She Said	Matrix: Reloaded Soundtrack - Mona Lisa	Nirvana - Smells like Teen Spirit	Goldfinger - Superman	Papa Rouch - Last Resort
99%	BPM	180.0122	136.88	173.679	167.978	90.52
	Confidence	7.625	12.041	6.387	4.686	6.38
	Time Taken	27.89	21.765	25.734	26.375	38.062
95%	BPM	179.987	136.971	115.875	194.888	181.196
	Confidence	11.615	20.268	13.3	9.99	10.581
	Time Taken	31.156	25.829	33	33.984	43.594
90%	BPM	180.0122	136.971	115.733	194.874	181.171
	Confidence	13.822	24.419	13.426	17.79	16.666
	Time Taken	37.407	32.125	43.5	52	49.156
1%	BPM	180.012	136.971	115.73	194.874	181.196
	Confidence	42.533	60.739	13.426	57.25	32.826
	Time Taken	129.656	171.297	145.36	151.844	153.328

The results above show that with a threshold as high a 95%, the correct BPM is still extracted. At 99%, errors in BPM calculation begin to appear. The lower the threshold, the greater the confidence in the BPM calculations.

In the final implementation, the threshold is set at 90%. This gives a nice balance between confidence and execution time.

7.1.4 Bar Start Discriminator

When dividing the song into bars, the beat on which each bar starts must be found. This is done by comparing self similarity graphs of each possible configuration of bar start positions, including having bars start on an offbeat.

Then some measure of the fitness of each bar starting position is made, and the fittest is taken as the correct bar start position. If this falls on an offbeat, the beat which follows it is taken as the bar starting beat.

Three measures of fitness were tested with five different songs.

The first measure is the simply the variance of the values in the self-similarity matrix. The second measure is the average difference between the values of neighbouring pixels. The third measure is the variance of the differences of between neighbouring pixels.

All songs in the test were normalised such that the first beat starts a bar.

	Move Your Feet	4 Ton Mantis	Afronova Primeval	Du Hast	Trip Machine Climax
Variance	**0.013583238**	**0.024740338**	0.014869877	0.015521936	**0.009694622**
	0.012447388	0.023243195	0.013620707	0.014978109	0.008617234
	0.012036733	0.022020121	0.013532147	0.014315723	0.008557323
	0.011702179	0.020068664	0.013350973	0.012979477	0.008667265
	0.011673491	0.019739749	0.013461003	0.013207228	0.009201143
	0.011693185	0.020949605	0.013953608	0.013262063	0.009065784
	0.01212741	0.022811938	0.014449666	0.013865416	0.008802133
	0.012781102	0.022992956	**0.014995169**	**0.016134458**	0.009091324
Mean of Absolute Diff	**0.064406974**	**0.07917991**	**0.048331542**	**0.063388439**	**0.063101395**
	0.060863599	0.076610093	0.042611761	0.061296777	0.058038202
	0.058144729	0.075458077	0.04232522	0.05834401	0.056746315
	0.053302514	0.066139713	0.041645215	0.05513148	0.059128004
	0.051427501	0.06522833	0.041771115	0.055593812	0.061687476
	0.048021686	0.069918	0.043931159	0.054772795	0.061438207
	0.047918943	0.074257622	0.044883676	0.056494581	0.059998968
	0.051542197	0.075443586	0.047810567	0.062807772	0.061330292
Variance of Absolute Diff	**0.003347478**	**0.007777727**	0.003271583	**0.006325854**	**0.003338439**
	0.002582028	0.006509342	0.002432732	0.005647775	0.002466287
	0.002235001	0.005207882	0.002466394	0.004874416	0.00245108
	0.002059288	0.003875608	0.002320258	0.00331446	0.002515684
	0.00198112	0.003686763	0.002361791	0.003172086	0.002773353
	0.001990746	0.004799728	0.003067769	0.003155723	0.002659963
	0.002006388	0.00660194	0.003543635	0.003423935	0.002382024
	0.002153141	0.00670415	**0.003782109**	0.005996017	0.002925888

All three of the fitness functions gave the correct results (in bold) in all five of the tested songs. Although the mean of the differences was the only one which did so without, in some cases, picking an offbeat.

The final implementation thus uses the mean of differences fitness function. This was found to provide a reliable estimate of the bar starting position in most songs encountered.

7.1.5 Size of Window for Gap Value Fitness Function

When determining the best BPM and gap value, the fitness function computes the fitness of each gap value based on the number of computed gap values found in a surrounding window.

The optimum size for this window was tested using five songs. For each song window sizes from 5 to 8000 samples were used. One sample is usually 1/44100th of a second.

Window Size in Samples		German Eurovision	Holly Valance - Kiss Kiss	Marilyn Manson - Rock Is Dead	Euromix - Dead End	Four Seasons (Techno Remix)
5	BPM	72.5488	96.99769	67.41401	131.26953	134.5059
	Gap in Seconds	2.22202	1.5307	0.81669	1.33574	0.15651
50	BPM	72.5488	96.99769	134.88301	95.01239	141.98326
	Gap in Seconds	2.22025	1.53073	0.84213	0.053832	0.25252
100	BPM	145.09761	96.99769	134.88301	190.00431	141.98326
	Gap in Seconds	0.56558	1.53073	0.84617	0.05283	0.25252
500	BPM	145.09761	193.98827	134.88301	190.00431	142.01374
	Gap in Seconds	0.56401	0.44893	0.84705	0.05283	0.26286
1000	BPM	145.09761	194.00259	134.88301	190.00431	141.9985
	Gap in Seconds	0.56247	0.454036	0.84757	0.05283	0.25624
2000	BPM	145.09761	194.01672	134.88989	190.00431	141.98326
	Gap in Seconds	0.56257	0.455737	0.84782	0.05283	0.25583
4000	BPM	145.08965	145.45654	134.89676	95.02262	70.96117
	Gap in Seconds	0.55857	0.46678	0.84916	0.06243	0.23651
8000	BPM	145.07374	145.45654	134.89676	95.05335	182.05587
	Gap in Seconds	0.5544	0.46678	0.84401	0.06458	0.25918

500 to 2000 Samples gave the correct results for all songs tested. Above and below these values errors started to appear.

For results where the BPM is correct, the same (or close enough) gap value is always found. This seems to suggest that the window used for deciding when gaps are similar is not of great importance beyond ensuring the correct BPM is found.

In the final implementation 1000 samples is the window size used. This should ensure the most accurate determination of BPM and gap value.

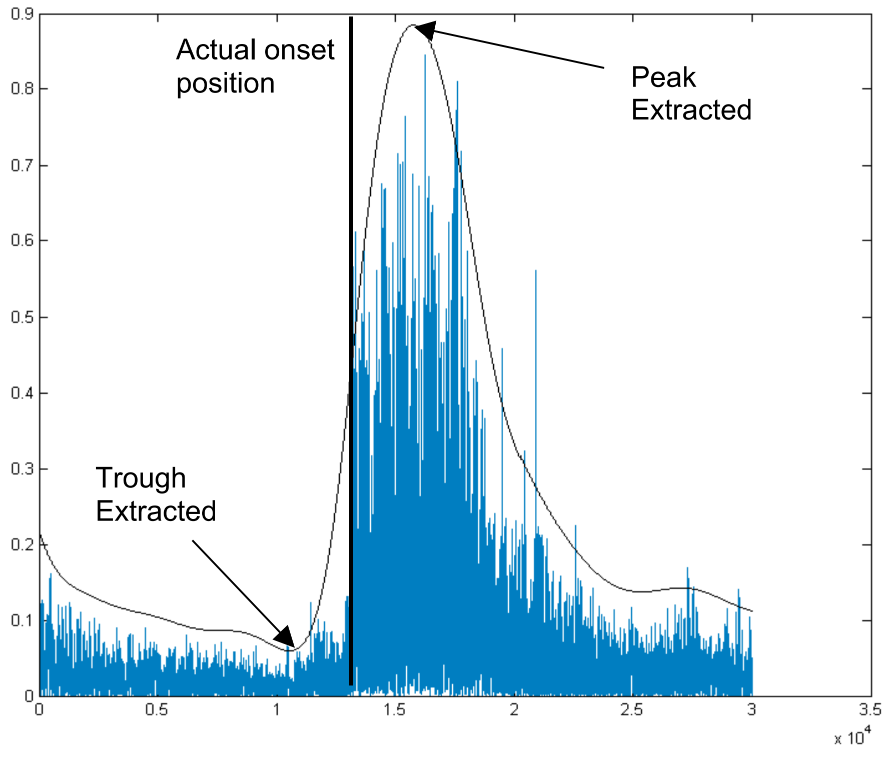


Figure 21 - Diagram showing the relation ship of the onset positions to the extracted troughs and peaks.

7.1.6 Threshold for Locating Exact Onsets

To find beats, peaks are picked from a smoothed representation of the waveform. But these peaks do not correspond exactly to the onset of the beat. The onset is the first occurrence of a high amplitude element in the waveform, and due to smoothing the peak never equates to this element.

In the smoothed waveform a trough, signalling the point in time before the beat occurs, precedes every peak. The correct onset will always be somewhere between the trough and its associated peak.

A section of the original waveform, from the location of each trough to each peak, can be examined to find the actual onset of the beat. This is considered to be the first sample in the section of the original waveform which is above some threshold.

The best value for this threshold was determined qualitatively. A range of songs were tried at different thresholds in order to find one which threshold would position the beats correctly for most songs. The results of each test were examined by hand using the StepMania simulator.

The threshold value was tweaked as new songs were tested. In the final implementation, the sample must be 70% of the height of the peak in the original waveform to be considered the onset.

Rather than find the onset for every beat separately, it was found that better results were obtained for BPM and gap value, if all onsets were considered to be the same distance from the peak they precede. This distance is determined by averaging the distance from of each peak from its detected onset.

7.2 Applicability of Algorithms

This section covers the tests performed to ascertain the applicability of certain algorithms being used.

7.2.1 Theoretical Worst Clique Finding Performance

The maximal clique finding algorithm used is that devised by Bron and Kerbosch. The implementation used had only previously been tested with graph sizes of up to 50 nodes. In this project the graph size varies between 50 to 120 nodes, depending on the BPM and length of the song.

It is important to test the applicability of the algorithm, as there are cases which could theoretically occur that are computationally infeasible. These cases will occur if the graph is too connected, or has too many nodes. High connectedness will occur in a song that is very similar all the way through.

120 nodes (each node corresponds to a bar) is the maximum that can occur, based on a song having the maximum BPM and being of maximal length. The test measured the performance of the algorithm on graphs of 50 to 120 nodes, with 50% to 90% of the nodes connected.

Number of Nodes	Connectedness of Graph				
	50%	60%	70%	80%	90%
50	0.125	0.391	1.265	4.328	15.422
60	0.562	0.953	2.734	17.063	80.625
70	0.938	1.906	7.75	70.172	-
80	1.546	3.641	17.328	202.453	-
90	2.375	7.219	57.062	-	-
100	3.813	12.516	106.015	-	-
110	5.359	21.719	-	-	-
120	8.281	35.984	-	-	-

The dashes in the above table indicate tests which did not finish within a reasonable time (5 minutes), or which ran out of memory.

The results show that there are cases that can occur, which are computationally intractable. Therefore, graphs which are intractable are detected before computation begins, and when found cause the application to terminate gracefully.

7.2.2 Typical Clique Finding Performance

This test measures the real world performance of the maximal clique algorithm on a selection of five songs.

	Junior Senior - Move Your Feet	Offspring - Living in Chaos	Scotty B - Drop The Bomb SF	Placebo - Taste in Men	A - Starbucks	Euromix 2 - Living in America
Connectedness	28.48%	7.92%	43.69%	28.51%	24.23%	57.94%
Number of Nodes	50	86	56	88	64	60
Time Taken	0.015	0.016	0.0309	0.094	0.469	0.032

The time taken to find all the maximal cliques in the tested songs is close enough to zero to be considered negligible. The Algorithm performs very well in practice, even though it is worst case performance can be very bad. This is mainly due to the low connectedness of the graphs created by this project.

Therefore it is unlikely that the application will ever be required to terminate due to an intractable graph.

7.2.3 The Best Representation of a Beat

In order to decide the best algorithm for computing the representation a beat, three different candidate methods were tested, and the quality of the self-similarity matrices they produced was were evaluated. The first candidate is the energy of each beat, calculated by taking the average of the square of each sample. The second candidate is the Mel Frequency Cepstral Coefficients (MFCC) vector for each beat. Finally, the last candidate is the Fourier transform of each beat, grouped into a discrete number of bins.

Five songs were tested with each candidate representation, and the resulting self-similarity matrices compared by hand. The best representation is the one providing the clearest visible division of each piece of music. In practice this turns out to be the energy representation. The MFCC representation consistently performs the worst, and generates uniform grey self-similarity matrices for songs that have clear divisions in structure. The Fourier transform and energy representations both generate clear similarity matrices, but the energy representation was chosen because it is faster to compute, and in practice it produced better grouping results.

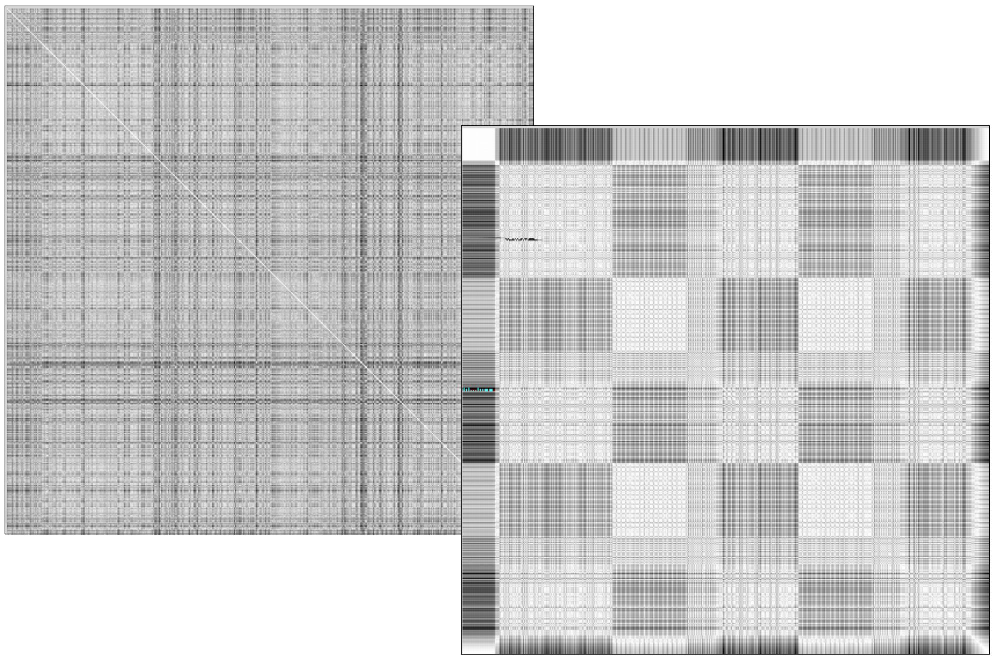


Figure 21 - The image on the left is the self-similarity matrix as generated by MFCC. The image on the right is that generated by the energy representation. It can be clearly seen that the energy representation provides a clear structuring to the song, where the MFCC representation fails to do so.

8 Evaluation

This section gives a critical evaluation of the finished project. It is divided into two parts, the first gives a quantitative evaluation of the BPM and gap value algorithm, and the second part gives a qualitative evaluation of all sections of the project.

8.1 Quantitative

8.1.1 BPM and Gap Value Test with ‘Official’ DDR Songs

In order to evaluate the quality of the BPM and gap value algorithm, two tests were used. In the first of these all the applicable songs from the Euromix 2 DDR arcade machine were used as inputs to the application, and the calculated BPMs and gap values were recorded. Applicable songs are those that do not change BPM part way through, and whose BPM is within the limits of the system (90 to 200 BPM).

The songs from the arcade do not have official step files detailing their BPM and gap values, but fans have recreated the step files by hand. In the tests these are used as the ‘official’ BPM and gap values, although due to being determined by hand they may still contain errors.

The Computed and ‘official’ BPMs are compared by simply taking the difference between the values for each song.

Song Name	‘Official' BPM	Computed BPM	Diff in BPM
Afronova Primeval	200	200	0
B4U	155.05	155.0905574	0.040557412
B4U - Oni Remix	170	169.974947	0.025052997
Burnin' the Floor	155.05	155.0632911	0.013291139
Candy	192.01	192.0034831	0.006516944
Can't Stop Falling In Love	155	155.0087873	0.008787346
Can't Stop Falling In Love - Speed Remix	170	169.9967877	0.003212335
D2R	160	159.9951627	0.004837344
Destiny	155	155.0087873	0.008787346
DXY!	148	148.0031323	0.003132341
Electro Tuned	125.08	125.0354409	0.044559115
Exotic Ethnic	190	189.9906656	0.009334386
Groove 2001	140	140	0
Higher	132.06	132.0820646	0.022064593
Hypnotic Crisis	134.97	135.0068881	0.036888107
Hysteria - Oni Remix	190.1	190.1271826	0.027182582
Let the Beat Hit 'Em	110	110.0162155	0.016215542

Living in America	168.98	169.0194826	0.039482593
Look to the Sky	140	140.0074078	0.007407799
My Summer Love	100.05	100.0794281	0.029428118
Never Ending Story	140	140	0
Never Gonna Make	136	135.9921879	0.007812098
Never Let You Down	125	125	0
Nori Nori Nori	160.3	160.3053435	0.005343511
Rain of Sorrow - Ballad 4 U	139.99	140	0.01
Secret Rendez-Vous	98	196	98
So Deep	139.97	140.0074078	0.037407799
Spin the Disc	130	129.9990174	0.000982608
Test My Best	147	147	0
The Centre of the Heart	128	128.0054182	0.005418219
Trip Machine Climax	180	180.0367422	0.036742192
WWW.Blonde Girl	170	169.9858666	0.014133368

The only song for which the incorrect BPM is calculated is Secret Rendez-Vous. In this case the application computed the BPM to be twice as fast as the actual BPM. Mistaking BPMs as being twice as fast, or half as fast as they actually are is a common problem, and one that plagues humans as much as computers.

The biggest difference otherwise is 0.045… of a second, for Electro Tuned. This is a small enough difference to be unnoticeable. Therefore the BPM algorithm can be considered very successful, with 31 out of 32 correct (96.9%).

The gap values in the ‘official’ step files, are given in 1000ths of a second. In order to compare them to the gap values generated by the program (which are in seconds), both must be normalised.

Both are normalised to the smallest possible gap value in seconds. This is simply the remainder of the gap value after dividing by the interval size of the BPM for that song.

Song Name	‘Official’ Gap value	Computed Gap Value	Normalised ‘official’ gap value	Normalised computed gap value	Difference in normalised values
Afronova Primeval	-1105	0.050839002	0.095	0.050839	0.044161
B4U	25	0.434829932	0.025	0.047959	0.022959
B4U - Oni Remix	685	1.0438322	0.332059	0.337846	0.005787
Burnin' the Floor	25	0.04170068	0.025	0.041701	0.016701
Candy	-1110	1.340997732	0.139935	0.09102	0.048914
Can't Stop Falling In Love	15	0.434716553	0.015	0.047642	0.032642

Can't Stop Falling In Love - Speed Remix	-1335	1.427210884	0.076765	0.01542	0.061345
D2R	370	0.393560091	0.37	0.018549	0.351451
Destiny	155	1.723945578	0.155	0.175646	0.020646
DXY!	100	1.74537415	0.1	0.123787	0.023787
Electro Tuned	255	0.468095238	0.255	0.468095	0.213095
Exotic Ethnic	5	0.027777778	0.005	0.027778	0.022778
Groove 2001	-1355	0.530362812	0.359286	0.101791	0.257494
Higher	20	1.861609977	0.02	0.044558	0.024558
Hypnotic Crisis	20	0.492743764	0.02	0.048322	0.028322
Hysteria - Oni Remix	-230	0.355578231	0.085623	0.04	0.045623
Let the Beat Hit 'Em	0	2.212063492	0	0.030567	0.030567
Living in America	20	0.398752834	0.02	0.043764	0.023764
Look to the Sky	-1260	0.841950113	0.025714	0.413401	0.387687
My Summer Love	20	1.236507937	0.02	0.03746	0.01746
Never Ending Story	-1640	0.038072562	0.074286	0.038073	0.036213
Never Gonna Make	55	0.946281179	0.055	0.063878	0.008878
Never Let You Down	325	1.778231293	0.325	0.338231	0.013231
Nori Nori Nori	30	0.798752834	0.03	0.050181	0.020181
Rain of Sorrow - Ballad 4 U	20	1.769750567	0.02	0.055465	0.035465
Secret Rendez-Vous	25	0.353945578	0.025	0.047823	0.022823
So Deep	20	0.485419501	0.02	0.056871	0.036871
Spin the Disc	950	0.501179138	0.026923	0.039637	0.012714
Test My Best	25	0.044421769	0.025	0.044422	0.019422
The Centre of the Heart	-1760	0.075986395	0.115	0.075986	0.039014
Trip Machine Climax	20	1.408231293	0.02	0.07517	0.05517
WWW.Blonde Girl	40	0.418185941	0.04	0.065215	0.025215

For the gap value calculations, there are only four songs for which the normalised gap values differ significantly. Two of these, Electro Tuned and Groove 2001, have been calculated incorrectly. The others, D2R and Look to the Sky, have been calculated correctly, the large difference is only an artefact of the normalisation process due to the gap value being very close to the interval length.

With 30 out of the 32 (93.8%) of the gap values computed correctly, the algorithm has proved to be very strong.

8.1.2 BPM and Gap Value Test with Non-DDR Songs

In order to test the real world applicability of the BPM and gap value algorithm, the application was tested with 25 well known songs, which have never been used before in DDR.

The songs chosen were considered to be good candidates for use in DDR. Therefore the test cannot be taken as proof of the power of the algorithm wih arbitrary music.

Without any ‘official’ BPM or gap values to compare with, an alternative rating strategy is used. Each song is run through the application, and its resulting step file is loaded into StepMania. An expert DDR player than danced to the arrow patterns produced by the application. If the arrows were consistently on time with the beats then BPM and gap algorithm was considered to work.

Song Name	Computed BPM	Computed gap value	Successful
The Faint - Agenda Suicide	144.5585664	0.259569161	Yes
Melanie C - I Turn to You	137.9705913	2.163356009	Yes
The Prodigy - No Good	145.002192	0.83446712	Yes
Pennywise - My Own Country	141.1953042	0.672222222	Yes
Nirvana - Smells Like Teen Spirit	115.7328435	0.916077098	Yes
BT - The Hip Hop Phenomenon	130.0437411	1.35260771	Yes
The Prodigy – Speedway	144.0313538	0.705328798	Yes
The Prodigy – Firestarter	141.4897599	0.438344671	Yes
Mclusky - Lightsaber Cocksucking Blues	137.8555799	0.848571429	No
A – Starbucks	150.0935958	0.020385488	Yes
Bomb the Bass - Bug Powder Dust	196.349065	0.681111111	Yes
Liam Lynch - United States of Whatever	149.2638348	1.122743764	No
Puretone - Addicted to Bass	164.9934526	0.67893424	No
Eminem - Without Me	112.2756397	1.479750567	Yes
The Cardigans - Erase Rewind	103.7118332	0.317006803	Yes
Girls Aloud - Sound of the Underground	163.9709983	0.328684807	No
Nickleback - This is How you Remind Me	171.8516594	0.59707483	Yes
Avril Lavigne - Sk8ter Boi	150.0156266	1.135791667	Yes
Holly Valance - Kiss Kiss	194.0024929	0.454036281	Yes
Avril Lavigne – Complicated	155.986559	0.690430839	Yes
Tatu - All the Things She Said	180.0122457	0.255510204	Yes
Infinity - Sunshine (When I Dance With You)	133.1655762	0.948367347	Yes
Red Hot Chili Peppers - By the Way	126.742348	1.419047619	No
Kylie Minogue - Spinning Around	120.0163288	0.722630385	Yes
Rammstein - Links 2 3 4	130.1076855	0.47952381	Yes

The program failed to find the correct BPM for five of the songs. In no cases did it incorrectly determine the gap value

Two of the songs it failed to find a correct BPM for, United States of Whatever and Lightsaber Cocksucking Blues, do not have a machine processed beat. Thus the drumming in each song wavers, produces inconsistent beat positions. Therefore there is not single BPM to be extracted and the program cannot be expected to find one.

Therefore the BPM and gap value algorithms are still very successful on real world songs. With an 80% success rate, the application is robust enough for public release.

8.2 Qualitative Evaluation

8.2.1 Placement of Pauses

The final implementation is very proficient at finding pauses where they exist; unfortunately it also tends to find pauses in places where they do not exist.

When the algorithm does find a correct pause, the effect is very noticeable and adds immensely to the feeling of an interesting and well-designed arrow pattern.

The extraneous pauses found by the implementation tend to suffer the problem that they have no defined start and end. They tend to occur in the middle of quiet sections of music. Therefore the player has not idea when to expect the arrows to start moving again. The whole effect seems rather arbitrary and nonsensical.

Fortunately these phantom pauses do not occur frequently and do not detract greatly from the overall arrow pattern.

8.2.2 Placement of Freeze Arrows

Due to time constraints near the end of the project, the algorithms used for finding and placing freeze arrows are both very simplistic. Even so, they successfully detect candidate positions for freeze arrows, and integrate these freezes into the existing arrow streams with little overall disruption.

The existence of these freeze arrows adds to the feeling of a well designed step track, as most new DDR songs contain some freeze arrows. Unfortunately there are some drawbacks to the simple methods used. The algorithm for detecting possible freeze positions does not allow for overlapping freezes, although these do occur in official DDR songs.

The main flaw is with the placement strategy; at present it does not attempt to take into account any of the rules for arrow flow. This means it is frequently the case (on higher difficulty levels) that freeze arrows are awkward to step on, and the arrows alongside the freeze cannot sensibly be followed. This problem, when it occurs, tends to make the step tracks look very amateurish.

Overall it is felt that, even with the drawbacks, the presence of the freeze arrows improves the quality of the generated step tracks.

8.2.3 ‘Flow’ of Arrows

Due to the rules determining which arrows can be placed depending on the position of the player’s feet, the final implementation is able to ensure that the arrows in the project flow in a natural manner.

Without any repeating sections, this flow is continuous throughout the whole song. Unfortunately, by grouping bars and thus introducing repeating sections this flow is broken. This is due to the generation of patterns for each group without knowing their positions in the song. Therefore it is not possible to track feet positions between bars.

This introduces breaks in the flow between bars of the song. This is usually only noticeable on harder difficulty levels, which have a greater density of arrows.

The broken flow turns out not to be a major problem. Due to the random nature of the arrow placement, and the fact that there are only four possible arrows, it turns out that in many cases there is no noticeable break in flow. Even in official songs there is almost never a continuous uninterrupted flow anyway. It is OK to have places in the song which require a nimble swapping of feet.

8.2.4 Grouping and Repeated Sections

The success of grouping, and thus repeating sections, was evaluated by watching arrow patterns along with the music and assessing whether the repeating patterns were really occurring on repeating sections of music. This assessment is very subjective, but is easy to perform, and gives a measure of the success of the grouping as perceived when the resulting step track is in use.

It was found that the grouping, while not optimal due to the use of a greedy algorithm for choosing the groups, works exceedingly well in practice. For every song tested it was found that the arrows repeated in ways which a human listener would agree with. Bars of music which did repeat, but didn’t have a repeating pattern arrow were not easily noticeable and so did not deter from the overall grouping effect.

8.2.5 Overall Evaluation

This section of the evaluation compares the performance of the project in relation to the state of the art.

The algorithm for extracting the BPM is very reliable and provides more accurate results than any existing system. Most state of the art BPM calculation algorithms have an error of 1% or greater in their results. The largest percentage error generated by the algorithm used in this project (discounting the case where double the BPM was extracted) is 0.037%. No existing system is even close to this kind of accuracy.

The algorithm is not perfect though; it is heavily constrained by the assumptions made about the input music. Its performance on a musical piece lacking a prominent bass line is likely to be poor. Its usefulness is also reduced somewhat by its non real-time nature, as much music analysis is performed in real time.

Finding the positions of the beats in the song accurately, as accomplished by extracting the BPM and gap value for a song, fills a small gap in current music analysis research. The algorithm developed by this project can provide, with extreme accuracy, the position of every beat in a piece of music.

Unfortunately this is not quite as impressive as it sounds. The assumption made, that the music analysed must have a strictly accurate beat severely reduces the usefulness of this algorithm in other systems. The algorithm is unable to predict the beat positions correctly for songs in which the BPM varies even slightly. This is not a problem for DDR, but in other systems is likely to be limiting.

The quality of the step tracks generated by the project are very high. In many ways they have the hallmarks and quality of the official step tracks. The system can reliably generate an interesting step track for every song, with some, of course, being better than others.

The main drawback of the system is the lack of originality in the step tracks generated. After experiencing a large number step tracks generated by this system, it is easy to spot that they all follow a similar theme. This is due to the same rules and heuristics being applied to every song; every step track ends up having the same feel.

Official step tracks break some rules, and provide interesting and unique features, due to their human creators not being bound by strict rules. Nonetheless the step tracks produced by this system are of a high enough quality that players are willing to return to the tracks to try and master them. In the fickle world of gaming this is the best compliment that can be hoped for.

The most promising aspect of this project is the development of an algorithm for grouping the music into repeated sections. The algorithm developed here can successfully construct a structural representation of any piece of polyphonic music (provided the music conforms to the assumptions made by this project). There is very little existing literature on this problem. The only research that touches upon this idea is An Automated Structural Analysis of Music [16] , but this has only been tested with three songs so far, and was unsuccessful at structuring one of these.

The grouping algorithm developed for this project, on the other hand, has been demonstrated to work with a large number of sample songs. It reliably divides the songs into a reasonable structure, which human listeners are happy with. It does not attempt to find the ‘correct’ structure of the music (if the music even has a correct structure), nor does it try and be 100% reliable in finding all structural elements. Under close scrutiny it can be seen that many of the groups made by the algorithm are non-optimal.

Nevertheless, the structuring extracted by the algorithm is consistently of a high quality. It has the potential to be useful in many areas other than DDR, for example providing structure to a light show based on music. The high reliability of the results means the algorithm is not just confined to music analysis research, but can easily be used in real world applications (as demonstrated by this project).

Finally, the finished application is of a high enough quality to be released publicly. The system performs reliably with most inputs, and degrades gracefully when unable to process the music.

9 Conclusion

9.1 Key Points

The project achieved all but one of the goals set in the specification (see appendix for the full specification). The only goal that was not achieved was the extraction of song information from ID3 tags. This is an unimportant feature, and has no real impact on the quality of the project.

The project has also demonstrated that state of the art in music analysis can reliably and efficiently be used in real world systems. The quality of the BPM and gap value algorithm shows that music analysis systems are not just the domain of research.

Many disparate areas of music analysis research have also been combined successfully. The integration of beat positioning, through BPM and gap value calculations, with music structuring, through self-similarity and clique finding, shows there is great benefits to be had with a combined approach to music analysis. Much research develops independent systems to perform specific tasks; the ability to share information between these tasks can improve the efficiency of all tasks involved.

The grouping algorithm that was developed has the potential to be useful in many other areas. Its reliability and results are way ahead of current systems.

This project also demonstrated the emerging creative power of computers. The ability of the project to create unique and interesting arrow patterns for each song shows that computers are now powerful enough to start producing their own creative works. Conversely the lack of originality, and the similarities that can be seen in all the computer generated step tracks, points out the striking limitations of the computers creativity.

Tying all of this together is the fact that the project has produced a fully functional piece of software, in a state where it can be released for public use.

9.2 Further work

Many interesting areas of further work are discussed in the extensions portion of the specification (see appendix). These points will not be covered here; instead opportunities for further work that have arisen from construction of the system are explored.

The most obvious area to be improved is the arrow generation, in order to give the computer more creative power. A system whereby rules and heuristics can be ignored in part would give the system the power to create unexpected and original works.

Alternatively, the use of a neural network for generating the step tracks could still be explored. Using a neural network should avoid accidental bias, which can occur when human knowledge is encoded as heuristics.

Further improvements to the feature extraction routines, could alleviate the similarity in step tracks. This could include extracting features such as vocal segments in a piece of music, or using an MFCC like representation of beats that can express more varied qualities of the music.

One extension proposed in the specification that is worth mentioning here, is the ability to process songs with variable BPM. This can include the case of a human drummer who will have inconsistent timings between beats, and the case of music that has sharp changes in speed.

This will require an overhaul of the current BPM algorithm, while still retaining its accuracy. Any misjudgements of the system with regard to the location of the speed changes will likely render the resulting step file unusable. Thus, this is a very difficult extension to implement, but which would remove the most restrictive assumptions made by the project.

Overall the project can be considered a great success, achieving all its goals, and contributing new and valuable work to the field of music analysis.

10 Bibliography

[1] DDR Freak website, http://www.ddrfreak.com/
[2] Dance With Intensity, http://dwi.ddrei.com/
[3] StepMania, http://www.stepmania.com/stepmania/
[4] Ogg Vorbis format, http://www.xiph.org/ogg/vorbis/
[5] Mp3 format, http://www.iis.fraunhofer.de/amm/techinf/layer3/
[6] dwi file format, http://dwi.ddrei.com/readme.html#4
[7] sm file format, Section 13 of http://www.stepmania.com/stepmania/readme.php
[8] bms file format, http://www.sun-inet.or.jp/~yaneurao/bm98/bmsformat.html
[9] DSP Guru website, http://www.dspguru.com/info/tutor/index.htm
[10] Tempo and Beat Analysis of Acoustic Musical Signals, Eric D. Scheirer, 1998, J. Acoust. Soc. Am. 103:1 (Jan 1998), pp 588-601, http://web.media.mit.edu/~eds/beat.pdf
[11] Beat Extraction from Digital Music, Will Archer Arentz, http://ilab0.ux.his.no/norsig/finalPapers/44.Beat_Extract_31820017132.ps
[12] The Beat Spectrum: A New Approach to Rhythm Analysis – Foote, Uchihashi, http://www.fxpal.com/people/foote/papers/icme2001.pdf
[13] Locating Singing Voice Segments within Music Signals, Berenzweig and Ellis, 2001, http://www.ee.columbia.edu/~dpwe/pubs/waspaa01-singing.pdf
[14] Foote, J., "Visualizing Music and Audio using Self-Similarity", in Proc. ACM Multimedia 99, Orlando, FL, pp. 70-80, http://www.fxpal.com/PapersAndAbstracts/papers/foo99b.pdf
[15] Riddim: A Rhythm Analysis and Decomposition Tool Based on Independent Subspace Analysis – Iroro Fred Onome Orife, http://eamusic.dartmouth.edu/~iroro/thesis/
[16] Listening to "Naima": An Automated Structural Analysis of Music from Recorded Audio - Roger B. Dannenberg, http://www.cs.cmu.edu/~rbd/papers/icmc02naima.pdf
[17] Introduction to Hidden Markov Models, University of Leeds, http://www.scs.leeds.ac.uk/scs-only/teaching-materials/HiddenMarkovModels/html_dev/main.html
[18] Project Proposal, Dance Dance Revolution, Lilian Chau and Nick White, MIT, 2002, http://www.mit.edu:8001/people/lec/DDR/ddr_proposal.pdf
[19] Digital Dancer, SigArch, University of Illinois, http://www.acm.uiuc.edu/sigarch/projects/dance/
[20] Audio Analysis using the Discrete Wavelet Transform – Tzanetakis, Essl, Cook, http://www.cs.princeton.edu/~gessl/papers/amta2001.pdf
[21] Online Introduction to DSP, Bores, http://www.bores.com/courses/intro/
[22] C. Bron and J. Kerbosch. Finding all cliques in an undirected graph. Communications of the ACM, 16:575--577, 1973.
[23] More information on MATLAB can be found at the MathWorks website, http://www.mathworks.com/
[24] LAME Mp3 encoder/decoder, http://lame.sourceforge.net/
[25] B. Logan, Mel Frequency Cepstral Coefficients for music modelling, http://ciir.cs.umass.edu/music2000, http://citeseer.nj.nec.com/logan00mel.html

11 Appendix

11.1 Introduction to Digital Signal Processing

Music can be represented as a signal of amplitude (strength of the signal, or volume of the music) changing with time. Amplitude can vary from -1 to 1, with values further from 0 indicating greater volume.

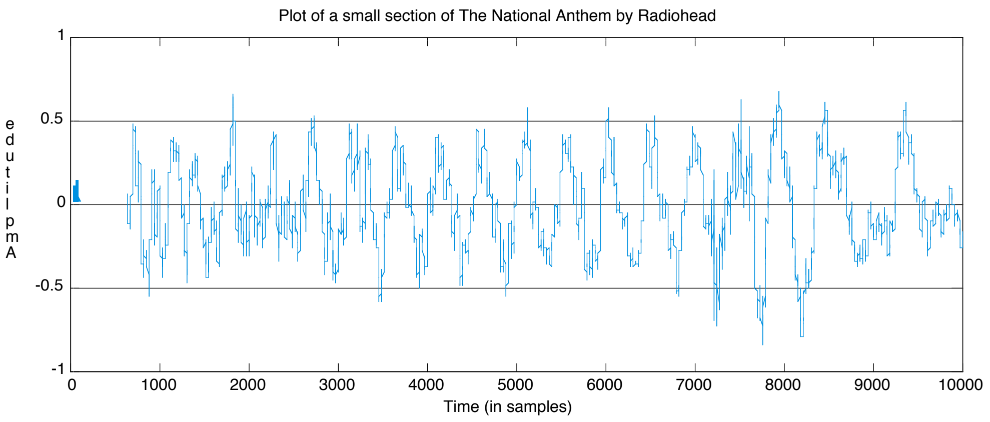


Figure 22 - A plot of a sample waveform. The high frequency elements give this plot its jagged look, while the low frequency elements give the overall repeating shape.

The frequency of a signal is the number of oscillations, changes in amplitude from positive to negative to positive again, per second.

Analog signals, such as sounds generated from a guitar, vary continuously with time. But in order to represent these signals in a computer we have to use a discrete time scale. To perform digital to analog conversion we sample the music at given intervals. This means we will miss any changes in amplitude between the sampling points.

This loss of signal data means that for given sampling rate, say 44100 samples per second, there is a maximum frequency we can accurately record in our digital signal. Nyquist worked out that the maximum frequency you can accurately record is half the sampling rate. In our case, about 22050 Hz, this is about the highest frequency perceivable by the human ear.

Sometimes the oscillating nature of the signal can be difficult to deal with. It can sometimes be more appropriate to deal with the amplitude of a signal on a purely positive scale. A simple process called rectification can be used to convert a signal to this form. Rectification is just the process of taking the absolute value of every sample.

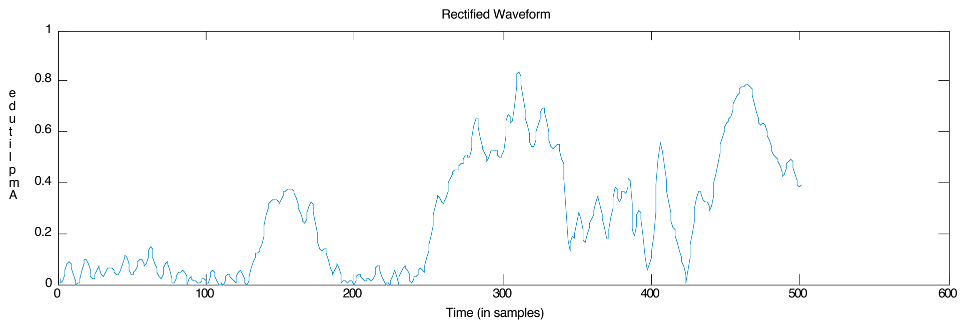


Figure 23 - A simple rectified waveform.

Once we have a digital representation of our signal there are many ways to analyse and manipulate it. Convolution is a useful, general purpose method of transforming a signal. It is defined as the weighted moving average of two signals, one of which has been reversed. What this usually means is that every sample in a signal is recomputed as a function of its neighbours, the parameters of the function being the second signal of the convolution process.

Convolution is a very computationally expensive process; every point in the original signal must be multiplied by every point in the convolution signal.

Convolution is the basis for digital filtering. A digital filter is simply a set of coefficients which define the amplitude of the convolution signal at each point. The number of coefficients is termed the order of the filter, so a filter with 6 coefficients is a 6th order filter. Due to the expense of convolution, digital filters are usually 10th order or lower.

Digital filters have many applications; one of the most basic is low-pass filtering. This is the process of removing all high frequency components from a signal. In music, high frequency components constitute the treble, while low frequencies constitute the bass. A low-pass filter can be used to remove all frequencies above bass level, leaving you with only the bass line of a song.

Band-pass and high-pass filters are an accompaniment to low-pass filters in extracting frequencies from a signal. Band-pass filters return you a signal only containing frequencies between a specified lower and upper bound. High-pass filters return a signal only containing frequencies above a certain bound.

Another useful signal analysis technique is the Fourier Transform. Jean Baptiste Fourier showed that any signal or waveform could be constructed by just adding together a series of sine waves of different frequencies with appropriate amplitude and phase. A computationally efficient way of determining the amplitude of the frequency components has been developed and is called a Fast Fourier Transform.

The result of a FFT is a power spectrum, showing the amount of each frequency contributing to a waveform. In essence the FFT converts a signal from a time/amplitude representation to a time/frequency representation. An inverse Fourier transform can be used to regain the original time/amplitude representation.

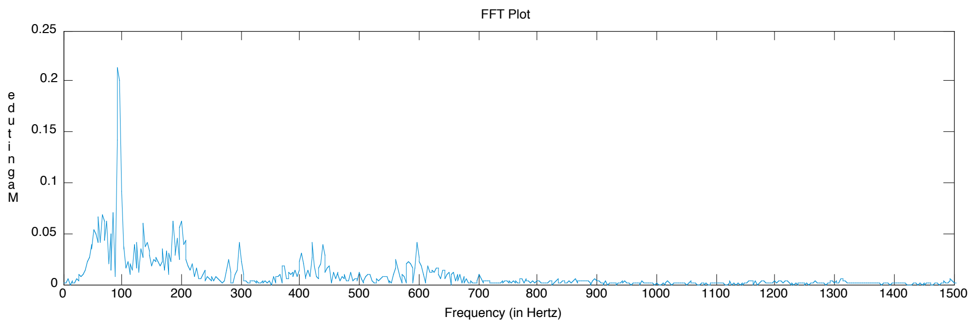


Figure 24 - The Fast Fourier Transform of the waveform given at the beginning of this section. It shows that the waveform has a very large frequency component of just under 100Hz.

Although the FFT is a very powerful tool, it was designed for use with stationary signals and gives very poor results with non-stationary signals such as music. In order to overcome this, the Short Term Fourier Transform was invented. It essentially applies the Fourier transform to small sections, or windows, of the signal at a time.

Wavelets and the Discrete Wavelet Transform (DWT) provide an even better alternative to the STFT. More information on the DWT can be found in "Audio Analysis using the Discrete Wavelet Transform" [20] .

For a more in-depth explanation of the main topics of DSP the reader is encouraged to look at the free online introduction to DSP from Bores [21] .

11.2 Specification

11.2.1 Overview of the Project

The aim of this project is to automatically generate step files for the PC DDR clones Dance With Intensity and StepMania. The system will work by accepting music files from the user, analysing them, and then outputting a step file which can be used by one of the DDR clones.

The project can be split into two major areas. The music analysis, to extract the BPM, Gap value and other salient details, and the arrow generation, which uses the features extracted from the music to generate fun and playable step files.

11.2.2 Core Specification

The system must be able to analyse music in the following file formats: .wav, .mp3.

The system must be able to accurately extract the BPM value from the music files to within 0.05 beats per minute. The system must be able to extract the Gap value correct to within 1 0.05 of a second.

DWI and SM use the BPM value to determine how far apart to space the arrows. If this spacing is not exactly the same as the spacing between beats in the music, the differences accumulate as the songs progresses. This means that a very small error in the BPM values can equate to arrows that completely fail to line up by the end of the song.

The system must ensure the BPM is calculated accurately enough such that the arrows do not drift with respect to the beats in the music. The Gap value must be accurate enough such that it is possible to obtain perfect ratings on the arrows through normal play.

The system is only required to deal with music files containing a prominent distinguishable beat. The BPM in the songs must stay constant throughout. The system is only expected to deal with music of a similar style to that used in arcade versions of DDR.

The style of music used in DDR always has a very prominent and simple beat, usually exaggerated compared to normal music. The average song only last about two minutes, which is shorter than normal music due to the physical exertion required. DDR music is mainly dance, techno or pop remixes, but any music with a loud steady beat can be used.

The system must be able to cope with music that contains quiet sections in which no beats occur, as well as those with sections in which off beat sounds are very loud. In these cases the system must still report the BPM and Gap value accurately.

The system must generate a fun and playable stream of arrows. This is a difficult metric to qualify. The arrows must be fun in that they are varied and challenging, but also playable in that it is actually physically possible to dance along to them.

The system must be able to output a step file in the following formats: .dwi, .sm.

When working with tagged music formats such as mp3s [5] , the system must transfer as much of the tag data as possible from the music file to the step file. The will usually include the title of the music and the artist performing it.

The user must be able to specify the location to which the step file should be saved. The system must be able to cope with directory structures of both DWI and SM when saving the file. The system must also have the option to copy the music to the same location to ensure the song is playable in DWI and SM without any other user intervention.

The system must be able to generate multiple difficulty levels per song, equivalent to the Basic, Trick, Maniac difficulty modes in DDR. Basic patterns should be between 1 to 6 feet, Trick patterns should be within 3 to 8 feet and Maniac patterns should be within 6 to 9 feet. The difficulty level in feet for the Basic mode must be less than the difficulty level of the Trick mode, which in turn, must be less than the difficulty level of the Maniac mode.

Most songs in DDR follow a similar sort of structure in the way they organise the patterns of arrows. Usually two or so base patterns, or motifs, of arrows are repeated, with slight variations throughout the song. One of these usually corresponds to the chorus section, and the other the verses. Although, these may not strictly be choruses and verses in the pop song sense of the words.

The variations in the patterns are usually manifested in the altering of the foot placement of part of the arrows. For example if the pattern consisted of alternating left/right arrows, it would be changed to alternating up/down arrows.

Freeze arrows are usually placed where notes are held for a period of time, making the player keep their feet steady in time with the music. In many cases while a single freeze arrow is held, other arrows still scroll up the screen. In this case the player must continue to hit the other steps with their free foot. In order to really get a feel for how the songs are structured the reader is encouraged to play DDR.

The different arrow patterns for a single song must all be of the same style such that the user can tell they are related. Generally each difficulty level should build upon the arrows of the previous difficulty, keeping the same overall structure. Small sections can be completely altered but the patterns should just be more complex versions of patterns in the easier difficulty levels. While attempting to stick to these guidelines the system must also ensure that all of the difficulty levels are fun and playable.

In DDR only some patterns of arrows are valid. For instance it is not possible to have more than two arrows reach the top of the screen at the same time.

Valid arrows are either single arrows or double jumps. Double jumps can consist of any combination of two arrows. Simple double jumps are the up/down and left/right pairs. Complex double jumps, or corner jumps, are the combinations of one vertical arrow with one horizontal arrow.

The timings of the arrows can be on the beat, known as a 1/4 note, because all DDR music is assumed to be in 4/4 time. Therefore every beat signals a quarter of a bar. Halfway between the beats, an 1/8 note. Other possibilities that are infrequently used include 1/12 note and 1/16 note. Other note positions are not valid.

For each pattern of arrows generated the system must also give the difficulty level of the pattern, measured in 1 – 9 feet scale of the arcades. The difficulty ratings assigned to the arrow patterns must be comparable to the arcade ratings.

- For songs of 1 feet, the arrows must be well spaced, usually occurring only once every third or fourth beat. Double jumps must be rare and only up-down or left-right. No offbeats are allowed.
- For songs of 2 feet the arrows may be spaced more closely.
- For songs of 3 feet the arrows may occur on every beat, but consecutive runs of arrows should not last long. Simple patterns of single arrows can occur. Simple double jumps can occur more often.

- For songs of 4 feet long periods of consecutive arrows usually occur. Offbeats may occur, but should be infrequent. Simple double jumps can occur very often.
- For songs of 5 feet offbeats may be a little more frequent. Complex doubles jumps, corner jumps, can occur often.
- For songs of 6 feet offbeats can occur frequently. Runs of offbeats should not be too longs and should have periods of rest in-between.
- Songs of 7 feet and above may contain any valid moves and should contain long complex patterns of arrows with many offbeats and complex double jumps.

Harder difficulty levels should contain more arrows per song than easier difficulty levels. A user of DWI has quantified the standard limits of number of arrows for each difficulty level [25]. These limits are shown below.

Level	Minimum	Maximum
1	1 step	106 steps
2	107 steps	136 steps
3	137 steps	170 steps
4	171 steps	194 steps
5	195 steps	228 steps
6	229 steps	262 steps
7	263 steps	305 steps
8	306 steps	366 steps
9	367 steps	∞

These rules can be altered slightly for songs with very low BPMs. In this case offbeat arrows appear more frequently and in much greater density.

The arrows that are generated must seem to ‘flow’ correctly. This means that while playing you should be able to follow the arrows in a manner where your body can flow smoothly with the arrows. This means there should not be forced to cross you legs or awkwardly swap feet just to hit all the arrows.

The system must be able to run on at least one of the same Windows platforms as DWI and StepMania. This would ideally be Windows XP. Support for other platforms should be included if possible.

It must be possible to run the program as a command line utility. The system must be able to run in a non-interactive mode to facilitate its use by other programs.

The system must have command line options that enable the user to:

- Specify which music file to generate a step file for.
- Specify the output type of the step file (dwi, sm).
- Specify which difficulty modes to produce patterns for.
- Specify, in feet, the difficulty level of each of the patterns.

- Specify the output location of the step file.
- Specify if the music file should also be copied to the same location as the step file.

The system must be able to generate a complete step file, including a different pattern for each difficulty mode, within a reasonable time. The system does not have to generate the patterns in real time. A reasonable time is defined to be roughly 10 minutes. This time limit is one of the least important areas of the specification, as the step files are likely to be generated only once, but played many times.

The system must be able to detect features of the song, such as areas that are quiet and areas that are very busy. The system should then use this information to produce arrow streams that reflect this; with quiet sections have only a few simple arrows and the busy sections containing complex and difficult arrow patterns. Other salient features of the music should be extracted as required to ensure that the generated arrow patterns are as good as possible.

11.2.3 Extensions

If there is time permitting, there are various extensions to the project that could be implemented to improve the quality of the step files produced by the system.

The system could allow the user to specify the difficulty level of the patterns generated as values on the groove radar. The user gives values for Voltage, Stream, Chaos, Freeze, and Air on a scale between 0 and 1. The system will then output a step file such that when the file is loaded in StepMania the groove radar displays values as close as possible to those given by the user.

The music analysis section of the system could be improved to cope with songs during which the BPM changes between distinct values. There are songs in existing mixes of DDR in which changes in beat occur. The system should be able to detect the correct BPM for each section as well as correctly determining the position of the change. The system does not need to cope with songs in which the beat changes rapidly, or many times.

The music analysis system could also be improved to detect pauses in the song. These pauses can be translated directly into pauses in the step file, where the arrows on the screen stop for the duration of the pause. There are songs in the existing DDR mixes during which the arrows stop moving when there is a pause in the song.

The arrow generation system could be improved so that it can generate arrow patterns for Doubles mode, where a single player dances using both pads. The generated arrow patterns should be in the same style as the patterns used in the arcade Doubles mode. It is important that the arrows flow between the two pads in such a way that it is possible for the user to hit all the arrows. This is important because the user now has to dance using both pads and it is very easy to create combinations of arrows that are physically impossible to dance to, such as the left arrow on the left pad followed immediately by the right arrow on the right pad.