Honours Project Dissertation

Digital Music Information Retrieval for Computer Games

Craig Jeffrey

University of Abertay Dundee
School of Arts, Media and Computer Games
BSc(Hons) Computer Games Technology
April 2016

Abstract

This project involves utilizing music information retrieval techniques to assist in the content creation process for music video games. The project aim was to explore how music information retrieval techniques could help content creators by automatically providing timing information and transcribing note locations. Tempo estimation is performed to automatically detect the beats-per-minute for a piece of music while onset detection is performed to transcribe the locations of notes. An application was developed which implements these two techniques in addition to a simple rhythm game in order to evaluate the generated gameplay. Objective evaluation of tempo estimation accuracy is performed through statistical analysis. The gameplay generated by onset detection is evaluated subjectively by playing the generated content and commenting on note placement. Results produced by the music information retrieval system were concluded to be suitable for use in assisting the content creation process. With further development, the system could be improved and expanded to be more reliable and useful to content creators.

Keywords: automatic music transcription, automatic content generation, game content creation, music information retrieval, onset detection, tempo estimation, BPM detection, beat detection, beat tracking.

Contents

List of Figures

List of Tables

1 Introduction

Music is indisputably a core component of modern day video games; it can help with immersion, setting the atmosphere or tone, and with emphasizing moments of significance - to mention only a few situations. Masterful use of music can elevate the experience of a game.

While important in most genres of video games, music video games are a genre of games which base their gameplay on the players interaction with music. The music video games genre covers many types of games with the most prominent of them being rhythm games where the core gameplay challenges the player's sense of rhythm. Many typical rhythm games require the player to simulate a real activity. A few notable games include *Dance Dance Revolution* (1998), where the player dances along to the music on a four-key dance mat, *Guitar Hero* (2005), where the player imitates playing guitar with a mock guitar controller, and *Beatmania* (1997), where the player imitates being a DJ using a controller with several keys and a turntable. There are also other rhythm games which are not directly analogous to real activities where the specifics of how to play along with the music is dependent on the game, e.g. *osu!* (2007), where the player aims and clicks circles on a computer screen in time with the music, or *Crypt of the NecroDancer* (2015), a roguelike dungeon crawler where the player's moves must match the beat of the music.

1.1 Beatmaps

In order to play along with the music in rhythm games, a file containing gameplay data is required. There is no common file format for these gameplay files; many games use different formats to suit their own needs, e.g. Beatmania's .bms format (Yane, 1998). For convenience this project will refer to these gameplay files as *Beatmaps* - the term *osu!* uses. The process of creating beatmaps is called *Beatmapping* (osu!wiki, 2016) and the person (or program) doing so a *Beatmapper*. A beatmap describes the gameplay component for a music track: it contains metadata about the music including beats-per-minute (BPM), offset value of the first beat of the first bar (from the beginning of the files sample data) and the position of all gameplay objects[1] for the game.

Beatmapping is a two step process. Figure 1 illustrates an overview of the beatmapping process.

[1] Gameplay objects refers to a rhythm game's game-specific objects that are synchronized with features of the music. Most rhythm games synchronize mainly with musical notes.

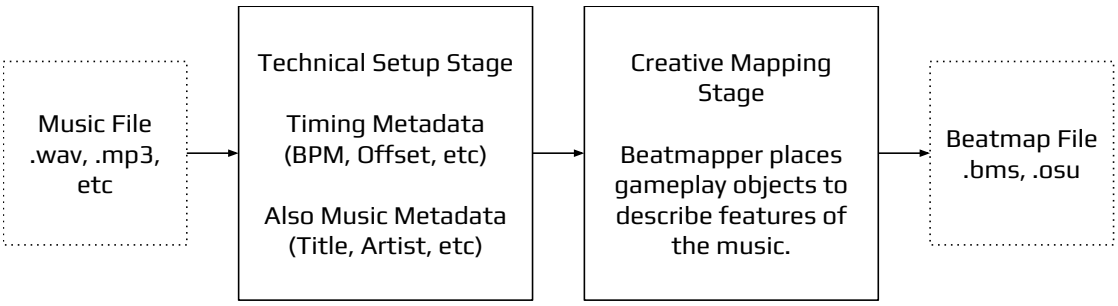


Figure 1: *Overview of Beatmapping*

The first step includes a timing process to find out the BPM and offset value - for several sections of the song if the tempo varies - so that gameplay objects placed by the beatmapper are consistent with the beat of the music. The timing process is time consuming and error prone, often requiring input from an experienced beatmapper to ensure that beats are sufficiently accurate. Inaccurate BPM leads to progressively worsening desychnronization between the music and gameplay objects whereas inaccurate offset leads to gameplay objects being consistently out of sync with the music.

The second step is the creative process of placing gameplay objects to describe features of the music. These objects do not necessarily correspond to notes on a musical score; where objects can be placed is often ambiguous and subjective to the person listening to the music. Beatmappers often have their own style of placing objects to describe the music, with the degree of creative freedom being limited only by the diversity allowed by each individual game's core mechanics.

This project proposes using Music Information Retrieval (MIR) techniques which involve automatically analysing music files to extract information to assist in the beatmapping process and content creation for other music video games by:

- Timing music, thereby providing suggested BPM and offset values and potentially lowering the experience required to begin beatmapping.
- Transcribing music features to help facilitate synchronization of game elements, e.g. gameplay objects, with music features, e.g. notes or beats.

Several games that automatically generate gameplay by analysing music files already exist such as *Audiosurf* (2008) where the player rides a three-lane track of the music collecting blocks in sync with the music. Additional examples of games exhibiting gameplay based on music features are *Beat Hazard* (2011) and *Soundodger+* (2013).

1.2 Aims and Objectives

To assist in defining the project goal a research question is posed:

> **How can Music Information Retrieval (MIR) techniques be used to aid in content creation for music video games?**

The project aims to develop a Music Information Retrieval system and a rhythm game to explore and evaluate the creation of gameplay using MIR techniques.

To achieve this aim, the project seeks to accomplish several objectives:

1. Research and implement MIR techniques to generate timing metadata and locate music features for arbitrary input music files.
2. Explore using the retrieved information for game content creation.
3. Evaluate the implemented MIR system, its application for content creation in music video games, and the resulting gameplay generated using the system.

2 Background and Literature Review

2.1 Music Information Retrieval

Music Information Retrieval (MIR) is a multifaceted field covering a number of sub-fields relating to retrieving information from music. One sub-field of MIR - Automatic Music Transcription (AMT) - can be described as the process of converting audio into a symbolic notation. This project will use onset detection to locate musical features and tempo estimation to determine the BPM of a piece of music. Both of these techniques rely on analysing an audio signal waveform.

2.1.1 Musical Features

This project uses the term "music feature" to describe the elements of music that are being timed and categorized for synchronization with elements of a game. The rationale for using this term is that it can be used as an umbrella term to describe several types of events present in a piece of music. Huron (2001) describes a feature as "a notable or characteristic part of something: a feature is something that helps to distinguish one thing from another". Using this definition, anything from entire sections of a song to a particular note could be considered a music feature. For this projects purposes only very specific features, i.e. features that can be localised to a specific point in time, are being referred to when mentioning music features. This mainly includes:

- Musical Notes - defined as a pitched sound.
- Musical Beats - defined as the basic unit of time for a piece of music.

2.1.2 Audio Signal Analysis

Audio files are typically stored as a finite number of samples in the time domain. When analysing audio signals it is often the case that analysing the time domain signal is only occasionally useful as the time domain only contains information about the amplitude of a signal (Bello et al., 2005). Many MIR techniques examine the signal in the frequency domain where the spectral content of a music signal can be analysed. Spectral content refers to the collection of frequencies present in a signal contributing to its frequency spectrum. To obtain a frequency domain signal from a time domain signal, the Short-Time Fourier Transform (STFT) is typically used (Bello et al., 2005; Dixon, 2006). The STFT uses a sliding-frame Fast Fourier Transform (FFT) at discrete points in time to produce a signal as a 2D matrix of frequency vs time. Essentially, an FFT produces a frequency window at a single point in time whereas the STFT adds a time

dimension. Figure 2 shows a signal's time domain and frequency domain representations for a single point in time (FFT window).

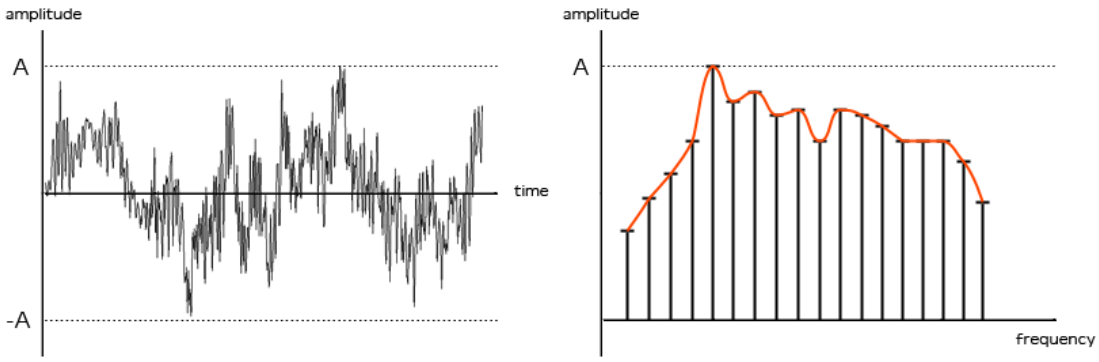


Figure 2: *Time Domain Waveform and Frequency Domain FFT Window*

Each line in the FFT window is referred to as a frequency bin. Frequency bins are discrete ranges on the frequency spectrum whereas a spectral envelope is a continuous curve in the frequency-amplitude plane which goes through each bin's peak, visually outlining the spectrum. The orange outline shows a potential spectral envelope.

2.2 Onset Detection

To find note locations, onset detection will be performed. Onset detection involves attempting to locate onsets present in a music signal. Bello et al. (2005) define an onset as the single instant chosen to mark the beginning of a note transient, where the transient is defined as a short interval in which the signal evolves in a non-trivial or unpredictable manner. Note that attack of a transient is not always a short sudden burst and may be a lengthy soft build up. Figure 3 illustrates an example of an onset with the signal waveform on the top and an annotated diagram of the onset on the bottom.

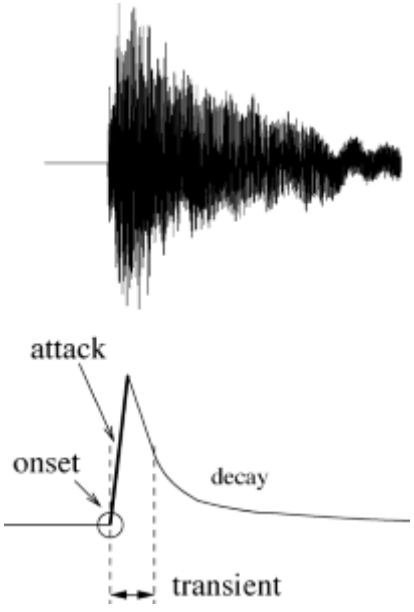


Figure 3: *Single Onset (Bello et al., 2005)*

Onset detection is a multi-stage process. The first stage is to optionally pre-process the signal to increase the performance of later stages. The next stage of onset detection is reduction of the audio signal using a detection function to emphasize points of potential onsets in the signal and then finally peak-picking to obtain individual onset timings. Figure 4 shows the process of a typical onset detector.

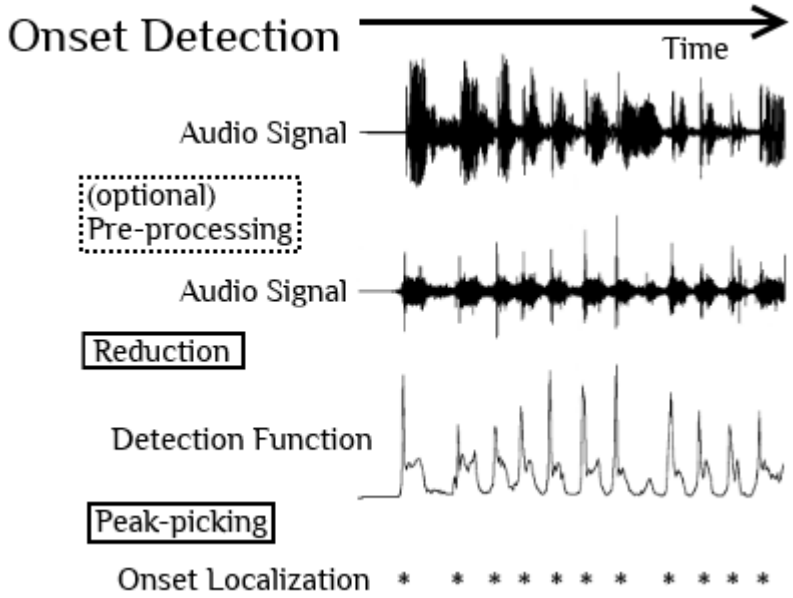


Figure 4: *Typical Onset Detector*

2.2.1 Onset Types

Bello et al. (2005), Dixon (2006) and Brossier (2007) distinguish between four onset types when evaluating detection functions. These are: pitched percussive (PP), e.g. piano or guitar; non-pitched percussive (NPP), e.g. drums; pitched non-percussive (PNP), e.g. violin and complex mixture (CM), e.g. a pop song. Onset types are distinguished because notes played on different instruments have different spectral envelopes which signify their presence in an audio signal. Most music used in games are likely to be complex mixtures where many types of onsets are present in the same signal. Bello et al. (2005) mentions that audio signals are additive and several sounds superimpose each other rather than concealing. Perfect onset detection is therefore incredibly difficult as many instruments have overlapping frequency ranges. Figure 5 shows these ranges for many instruments.

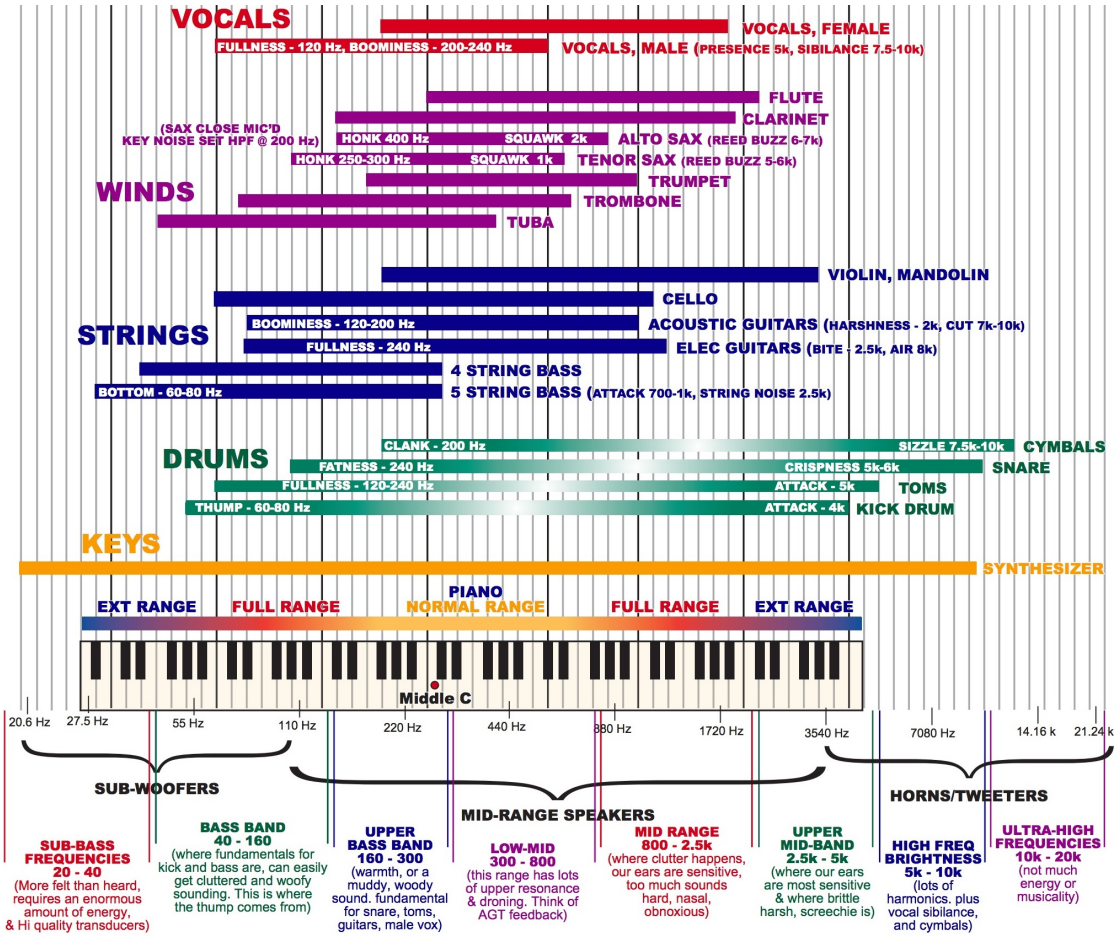


Figure 5: *Instruments by Frequency (Carter, 2003)*

Further classification can be done by introducing music texture, which can be briefly defined as the way in which melodic, harmonic and rhythmic materials are combined in a piece of music. This project distinguishes between monophonic texture - where a piece has a single melodic line with no accompaniment, e.g. most solo instruments - and polyphonic music texture - where a piece has more than a single melodic line, e.g. multiple independent melodies, accompaniment or any other variations which are not monophonic.

MIREX (2016) provides benchmarks for many MIR tasks in the form of a yearly competition. When looking at *Onset Detection Results per Class* from MIREX (2015a) it is clear that some types of onsets are more difficult to analyse than others. MIREX categorizes audio files into 4 classes:

- Solo drums (NPP)
- Solo monophonic pitched including 6 sub-classes:
  - Brass (PNP)
  - Winds (PNP)
  - Sustained strings (PNP)

  - Plucked strings (PP)
  - Bars and bells (PP)
  - Singing voice (PNP)
- Solo polyphonic pitched - Mostly PP as this covers instruments that can produce multiple simultaneous melodies such as piano, harpsicord and electric keyboard, however this could potentially include polyphonic PNP instruments as well.
- Complex mixtures (CM)

Class	BK7	CC6	CC7	CS2	LSY1	SB2	SB3	SB4	SB5
Complex	75.9001	69.8131	79.5290	60.4519	74.8342	79.4093	77.1321	77.5436	75.7033
Poly_Pitched	91.1332	81.1789	93.8976	78.4882	87.1040	94.0652	93.2909	91.5858	91.6566
Solo_Bars_And_Bells	97.2222	80.3567	100.0000	97.2222	95.5556	100.0000	100.0000	96.6667	96.4989
Solo_Brass	76.3213	62.2026	76.9750	66.4357	77.5745	82.1369	78.9370	76.6499	75.3497
Solo_Drum	92.8095	89.3099	93.0910	87.3415	90.9617	93.0738	93.7478	92.3994	93.0521
Solo_Plucked_Strings	87.9581	72.9415	91.4557	76.5770	89.8516	90.9030	92.3409	89.8442	89.7095
Solo_Singing_Voice	59.3605	17.4294	55.3185	21.3953	39.8520	52.1085	49.6470	60.6356	60.4122
Solo_Sustained_Strings	62.8483	52.8066	66.9198	12.1893	66.9673	72.9336	67.8520	58.7953	57.5217
Solo_Winds	68.2363	49.6935	72.2324	11.3785	71.0047	73.9666	71.2611	68.5523	74.5718

Figure 6: *Onset Detection Results F-Measure per Class (MIREX, 2015a)*

Figure 6 shows F-Measure[2] per Class onset detection results for MIREX 2015. In general it appears that PNP onset types - including singing, wind instruments and sustained strings - are the most difficult to analyse - even more so than complex mixtures. The second worst results appears to be, as expected, complex mixtures where many onset types are present. Percussive signals have overall better detection results. This is likely due to percussive instruments usually producing sharp attacks and short transients whereas non-percussive instruments generally produce soft and potentially lengthy transients. For these instruments, onsets may not be localized accurately to a specific point in time even if correctly detected.

2.2.2 Preprocessing

Preprocessing may be done in order to achieve specific results - such as detecting particular onset types - or to simply improve the results of onset detection. Bello et al. (2005) discusses several scenarios where others have split a signal into

[2]F-Measure is a method of determining accuracy using precision and recall where precision is a measure of relevant detected onsets and recall is a measure of relevant onsets that are detected.

multiple frequency sub-bands using filter banks. Splitting a signal into sub-bands implicitly categorizes onsets into frequency ranges and may be useful for game content generation if a game is looking to synchronize with particular instruments or frequency bands.

2.2.3 Detection Functions

Detection functions are the core component of an onset detector, they are used to process an input signal into a form where peaks of the signal waveform indicate onsets. Many detection functions exist with different strength and weaknesses depending on the signal type. A summary of the functions reviewed in literature in addition to the strengths and weaknesses of each one is given below. "+" denotes a strength whereas "–" denotes a weakness. Implementation details are kept brief as the specifics vary between individual implementations. The implementations used by this project will be discussed in the methodology section later.

1. **Time Domain Magnitude (a.k.a Attack Envelope or Energy Based)**
   A simple early method of onset detection discussed by Marsi (1996) which involves following the amplitude of the input audio signal.

   + Computationally fast as the function operates in the time domain thus no STFTs have to be performed.
   + Accurate at time localization of onsets due to processing being performed on individual samples instead of STFT windows.
   + Can be effective on monophonic signals or signals where the amplitude indicates onsets clearly, i.e. solo percussive. (Bello et al., 2005)
   – Limited usefulness because of only analysing time domain of a signal. Ineffective on polyphonic music where signal amplitude is not enough information to reliably locate onsets. (Bello et al., 2005)

2. **High Frequency Content (HFC)**
   A magnitude-based frequency domain function proposed by Marsi (1996) which involves detecting changes in the frequency spectrum between STFT windows. Frequency bin magnitudes are multiplied proportional to their frequency (hence the name, since higher frequency bins are factored more) and added together.

   + Good performance detecting percussive signals since they are generally indicated by broadband bursts across the frequency spectrum which are emphasized by this function. (Bello et al., 2005; Brossier, 2007)

+ Can work reasonably well on complex mixtures when percussion is present. (Bello et al., 2005; Brossier, 2007)

– Poor performance on non-percussive signals and onsets with relatively little energy. (Bello et al., 2005; Brossier, 2007)

3. **Spectral Difference (a.k.a. Spectral Flux)**
   Also described by Marsi (1996), this function involves measuring the change in magnitude of each frequency bin between STFT windows. Positive differences are summed and onsets are indicated by large differences (Dixon, 2006). Bello et al. (2005) mentions that Foote's (2000) method can be an alternative way of implementing a spectral difference function using self-similarity matrices. Two variations of the spectral flux function introduced by Böck, Krebs, and Schedl (2012) and Böck and Widmer (2013a) are among the current best performing detection functions. These are labelled as BK7 and SB4 respectively (Böck et al., 2015) on the MIREX 2015 Onset Detection Results in Figure 6 above.

   + Appears to perform reasonably on all types of signals. (Bello et al., 2005; Dixon, 2006; MIREX, 2015a)

   + Bello et al. (2005) recommends this function as a good choice in general.

4. **Phase Deviation**
   So far functions have only used magnitude information. Since for non-changing signals the phase distribution is expected to be near constant (Bello et al., 2004), onsets can be detected by comparing the change of phase in each frequency bin between STFT windows. Dixon (2006) improved the phase deviation function by weighting the phase deviation values of each frequency bin by their corresponding magnitude to eliminate unwanted noise from components with no significant contribution.

   + Good performance for pitched signals. (Bello et al., 2004, 2005; Dixon, 2006)

   – Poor performance on non-pitched signals. (Bello et al., 2004, 2005; Dixon, 2006)

   – Poorer performance on complex mixtures than other functions. (Bello et al., 2004, 2005) In Dixon's (2006) improved results performance was only marginally worse than Spectral Flux and Complex Domain on complex mixtures.

5. **Complex Domain**
   The complex domain function combines amplitude and phase information between STFT windows by comparing the amplitude and rate of phase change

using a distance measurement (Dixon, 2006). This function could potentially be considered a combination of spectral difference and phase deviation. A variation of the complex domain function introduced by Böck and Widmer (2013b) is among the best performing results for MIREX 2015. It is labelled as SB5 in Figure 6 above. (Böck et al., 2015)

+ Reasonable performance on all signal types. (Bello et al., 2004; Dixon, 2006)
– Slightly under-performs other functions on their specialised onset types. (Dixon, 2006; Brossier, 2007)
– In the results of Bello et al. (2004) the complex domain function outperforms spectral flux but in all more recent results the opposite is true in almost every situation.
– Slightly more computationally expensive than the other algorithms. (Bello et al., 2004)

6. **Recurrent Neural Network**
The current state-of-the-art performing onset detection function (MIREX, 2015a) is an artificial intelligence (AI) trained to recognize the locations of onsets using a neural network. The two best performing functions at MIREX (2015a) were an offline, non-real-time implementation (Eyben, Böck and Schuller, 2010) and an online real-time implementation (Böck et al., 2012) of this function respectively labelled SB2 and SB3 on Figure 6. (Böck et al., 2015)

+ Good performance on all types of music since the neural network learns where onsets typically occur from training data. (Eyben, Böck and Schuller, 2010).
+ Performs on par with or better than all of the above functions. (Eyben, Böck and Schuller, 2010; Böck et al., 2012; MIREX, 2015a)
– The function must be trained using a data set. This is time consuming and requires a data set appropriate for the type of music that it is planned for the function to be used on.

This is not an exhaustive list and other functions such as a statistical probability function covered by Bello et al. (2005) can also be effective. The functions listed are in chronological order of their introduction to the covered literature, including the most common functions and the function that can be considered state-of-the-art. Results can potentially be improved by combining functions such as the dual HFC x Complex method discussed by Brossier (2007) which was shown to have superior results in many cases to single functions.

2.2.4 Peak-Picking

After the detection function has reduced a signal, peak-picking is done to obtain individual onsets. Optional post-processing can be done to improve peak picking such as using a smoothing function to reduce noise (Bello et al., 2005). Thresholding is performed to determine a cut-off point for picking onsets. Peak-picking is then finalised by recording every peak above the threshold.

2.3 Tempo Estimation

Tempo estimation is the process of attempting to extract the tempo of a piece of music. Tempo is defined as the speed of a piece of music and is measured in beats per minute (BPM). A large number of approaches exist that achieve tempo estimation through varying methods. Zapata and Gómez (2011) evaluate a large number of tempo estimation methods. Figure 7 shows an illustration of the general scheme of tempo estimation methods with block descriptions below.

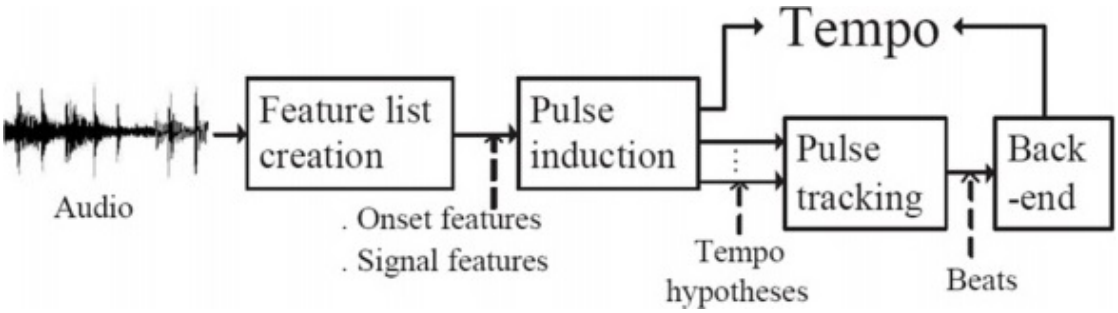


Figure 7: *General Scheme of Tempo Estimation (Zapata and Gómez, 2011)*

- **Feature List Creation**
  Transformation of the audio waveform into features such as onsets.

- **Pulse Induction**
  Use of the feature list to estimate tempo.

- **Pulse/Beat Tracking**
  Locates position of beats, potentially using already detected features (beat tracking is essentially specialized onset detection). Similar to feature list stage except beats are more relevant than features for tempo estimation.

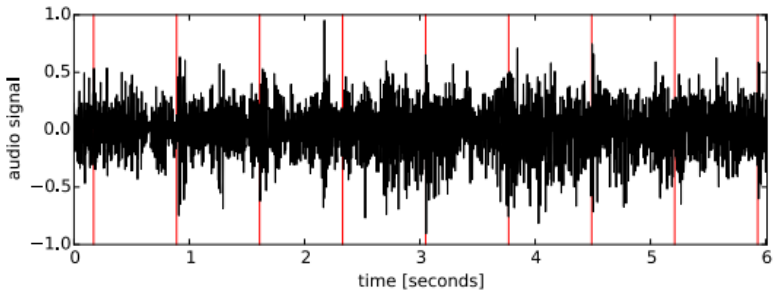
- **Back-end**
  Uses beat positions to estimate tempo or selects strongest tempo from current candidates.

Some methods don't include the third and fourth blocks as they simply use onsets or other features rather than performing beat tracking to estimate tempo.

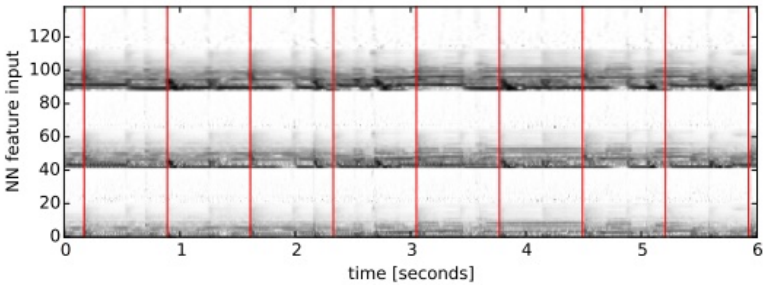
Evaluation of tempo estimation methods is much simpler than onset detection functions as the tempo is either correct or not. Note, however, that there can be multiple "correct" tempos for a piece as multiples of the true tempo have beats occurring concurrently, e.g. a song of 200BPM will have beats occurring at the same time interval twice as often as a song of 100BPM.

Given that there are many tempo estimation methods but only one evaluation criteria it is not worth reviewing a large number of tempo estimation methods. Their applicability to different types of music is deferrable to the onset detection or beat tracking functions that they are based on rather than their methodology of obtaining tempo from these features.

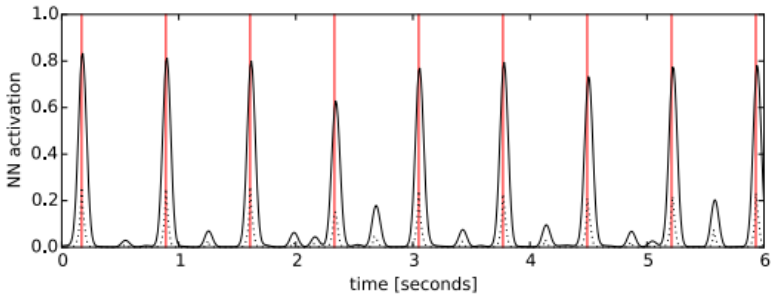
The current state-of-the-art tempo estimation method according to MIREX (2015b) *Audio Tempo Extraction results* developed by Böck, Krebs and Widmer (2015) is based on a neural network to determine which frames are beats and then a resonating comb filter bank to process beats and obtain tempo estimates which are then recorded on a histogram. The highest peak on the histogram is then selected as the tempo estimate when processing is completed. Figure 8 illustrates the process of this function visually.



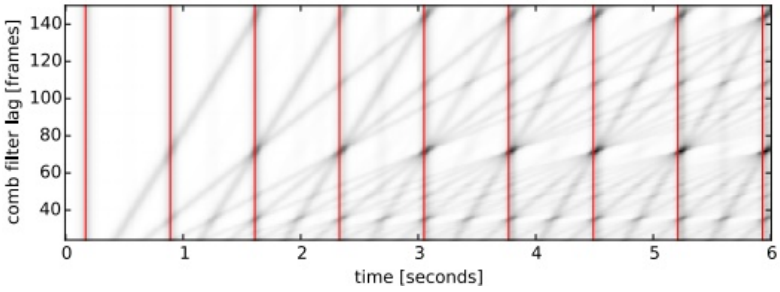
(a) *Input Audio Signal*



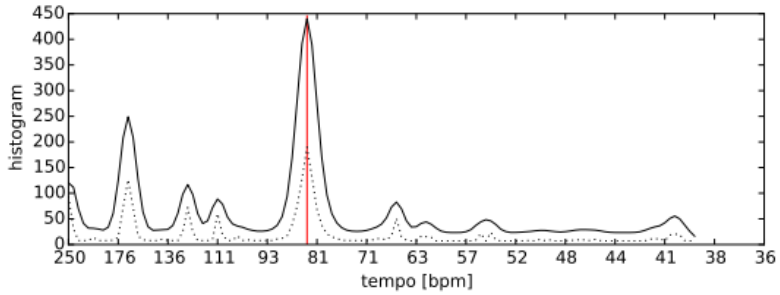
(b) *Neural Network Input*



(c) *Neural Network Output*



(d) *Resonating Comb Filter Bank Output*



(e) *Histogram of Tempo Estimates*

Figure 8: *Tempo Estimation based on Recurrent Neural Network and Resonating Comb Filter (Böck, Krebs and Widmer, 2015)*

2.4 Existing Tools

The project *"Dancing Monkeys"* by O'Keeffe (2003) generates step files (i.e. beatmaps) for *Dance Dance Revolution* (1998) using an independently developed implementation of Arentz' (2001) beat extraction algorithm to calculate BPM and a self-similarity matrix, as described by Foote (1999), to place the same generated step patterns in similar parts of a song. O'Keeffe was able to accurately determine BPM within ±0.1 of the correct BPM for a constrained set of input music. This accuracy was achieved by making assumptions about the input music - namely that it should have consistently occurring beats (i.e. computer generated) and a single tempo. O'Keeffe notes that the gameplay generated by the computer lacks originality, mentioning that official *Dance Dance Revolution* step files often break some rules to make gameplay interesting. In his evaluation, O'Keeffe also mentions that the structural analysis performed

to place note patterns is not objectively "correct" or even optimal as that is not what is attempted. What matters more is that the output is reasonable and generates agreeable gameplay.

2.5 Research Conclusion

Benetos et al. (2012) provides an insightful overview of the state of automatic music transcription, mentioning that the methods available at the time converge towards a level of performance not satisfactory for all uses. The most important takeaway is the notion that better results can generally be achieved by providing more input information about a music piece, e.g. the genre of the music or instruments used, so that the most effective methods and parameters can be used. Several important points and ideas can be summarized from research findings which will guide development of the project's MIR system:

- Many onset detection functions exist that excel at detecting different onset types, i.e. notes played by different instruments. This can be taken advantage of by using functions suited to particular music signal types. However more recent methods such as the neural network function are more universal in their effectiveness (Eyben, F., Böck, S., Schuller, B., 2010), i.e. they are simply better than their predecessors if their usage conditions are met.

- To automatically categorise onsets by their frequency, onset detection can be performed on frequency sub-bands of a piece of music.

- In conjunction with using a function suited to a particular type of onset, categorising onsets by their frequency may be useful to attempt onset detection for particular instruments. For example, to discover onsets for notes played on different piano keys, using a detection function suited to pitched percussive onsets on the frequency bands associated with each key could be attempted.

3 Methodology

This section provides an overview of completed practical work including details of the developed system for beatmap generation and rhythm game to be used for exploring the application of Music Information Retrieval to games. The application developed - dubbed *RhythMIR* - includes several features to enable the exploration of creating gameplay using onset detection and tempo estimation. In order to evaluate creating gameplay effectively, three main systems were developed for *RhythMIR*. These are: the **beatmap generator** including tempo estimation and onset detection, for generating gameplay files; the **rhythm game**, for testing gameplay created using generated beatmaps; the **filesystem**, for saving and loading beatmaps so that generation does not need to be repeated for every play session.

3.1 Beatmap Generation

When beginning application development, the decision to use a third party library for MIR tasks was made. Doing this allowed for more freedom in exploring gameplay creation by using several methods of onset detection rather than individually implementing a single method. The library that was chosen to perform MIR tasks is *aubio* (2015). This is because it is written in C and proved easy to integrate into a C++ application while providing many facilities including the choice of several onset detection functions.

To begin beatmap generation there must be at least one song available in RhythMIR to use as the source. The beatmap must be given a name then the generation process can be started. The generation process produces different output depending on a number of settings shown in the Generation Settings Window in Figure 9. Settings are explained throughout this section.

3.1.1 Setup

At the beginning of the beatmap generation process, a new **std::thread** is started to begin processing the audio file. Processing is executed on a separate thread so that the application does not block while processing. An **aubio_source_t** object is created to load in the audio samples from the source audio file. The source object takes two parameters, the song sample rate and the hop size. Sample rate is the amount of samples per second in an audio signal measured in hertz (Hz). Hop size is the amount of samples to advance every frame of processing. The amount of time, in seconds, for each hop can be

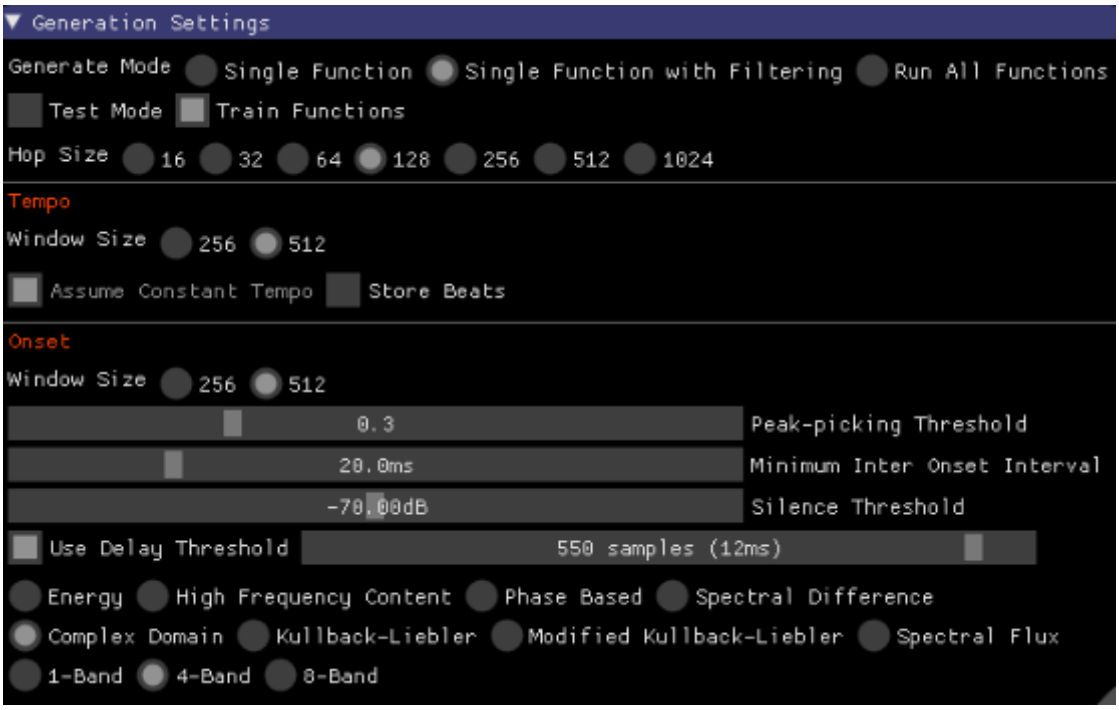


Figure 9: *Generation Settings Window*

calculated as $t = \frac{hopsize}{samplerate}$. Smaller hop sizes increase the time resolution of onset detection (and beat tracking for tempo estimation), allowing onsets to be distinguished closer together. Lower hop size therefore means more detections at the cost of more computation time.

After the source object has been set up, the **aubio_tempo_t** and **aubio_onset_t** objects are set up depending on the selected Generate Mode (Figure 9) which will produce one of three beatmap types. The beatmap types are:

Single Beatmap with a single note queue.

Four Key Beatmap with four note queues.

Visualization Beatmap not intended for playing, containing any number of note queues.

The three generation modes are:

Single Function A single onset object is set up using a single onset detection function, which will produce a single queue of onsets. Generated beatmap type is **Single**.

Single Function with Filtering 1, 4 or 8 onset objects are set up each using identical onset detection functions. Produces 1, 4 or 8 queues of onsets. Generated beatmap type is **Single**, **Four Key** or **Visualization** depending on number of bands selected.

Run All Functions 8 onset objects are set up - one for each onset function. Produces 8 queues of onsets. Generated beatmap type is **Visualization**.

The **aubio_tempo_t** object takes four parameters to set up: the name of the onset detection function to use for beat tracking (only option is default), the Fast Fourier Transform (FFT) window size in sample count, hop size in sample count and the signal sample rate in Hz. The onset detection function used for beat tracking is an implementation of the spectral flux onset function described by Dixon (2006), discussed above in the Detection Functions section (2.2.3).

Similarly, the **aubio_onset_t** object takes the same four parameters except the first parameter has a number of options for different onset detection functions. An overview of function strengths and weaknesses shown by others is discussed above in the Detection Functions section (2.2.3) for all functions except KL and MKL. The available onset detection functions include:

Energy Calculates local energy on the input spectral frame, similar to the Time Domain Magnitude function discussed before but using magnitude across frequency spectra instead of in the time domain.

High Frequency Content (HFC) Linearly weights the magnitude of frequency bins across the FFT window, emphasizing broadband noise bursts as onsets. Based on the HFC function in Marsi's thesis (1996).

Complex Domain (CD) A complex domain function implemented using the euclidean (straight line) distance function to emphasize large differences in both magnitude and phase between FFT windows as onsets. Based on the Duxbury et al. (2003) paper.

Phase Deviation (PD) A phase based function which emphasizes instability of the phase of the audio signal in each frequency bin as tonal onsets. Implementation based on the Bello and Sandler (2003) paper.

Spectral Difference (SD) A spectral difference function which emphasizes the difference in spectral magnitudes across FFT windows as onsets. Implementation based on the Foote and Uchihashi (2001) paper.

Spectral Flux (SF) A spectral flux function similar to the SD function above. Implementation based on the Dixon (2006) paper.

Kullback-Liebler (KL) A type of complex domain function using a logarithmic distance function, ignoring decreases in the spectral magnitude.

Due to the logarithmic nature of the function, large differences in energy are emphasized while small ones are inhibited. Based on a paper by Hainsworth and Macleod (2003).

Modifier Kullback-Liebler (MKL) A variation of the KL function described by Brossier (2007) which removes weighting of the current frames outside of the distance calculation, accentuating magnitude changes more.

The strengths and weaknesses of each function in addition to their applicability for gameplay generation will be discussed in the results section.
In addition to the above mandatory setup, four additional parameters are used to control the behaviour of onset detection. These are briefly explained below:

- **Peak-picking Threshold** - changes the cutoff threshold for labelling onsets on the reduced signal, higher threshold causes less onsets.
- **Minimum Inter-Onset-Interval** - changes the minimum amount of time (in ms) between when onsets can be detected.
- **Silence Threshold** - changes the relative loudness threshold (in dB) for determining silence.
- **Delay Threshold** - amount of time (in ms) to subtract from detected onsets to fix delay caused by phase vocoding (phase vocoding is explained in the processing section).

3.1.2 Processing

After setup is complete, the processing of the audio file begins. Hop size number of samples are read by the source object into a source buffer each loop iteration until processing is cancelled or there is not enough samples remaining to perform another hop. Listing 1 shows pseudo-code for the processing stage.

Listing 1: *Processing Stage Pseudocode*

```
1  while not canceling generation and frames were read last loop
2      aubio_source_do - read from source to source buffer
3      aubio_tempo_do on source buffer
4      if a beat was found
5          add the estimated BPM to BPMs vector
6          add the beat to the beats vector
7          if storing beats in beatmap (Figure 9)
8              add the beat to the beatmap beats vector
9      if not using filters
10         for all onset objects
```

```
            aubio_onset_do on source buffer
            if an onset was found
                add it to the note beatmaps vector
    else we are using filters
        filter from source buffer into filter buffers
        for all onset objects
            aubio_onset_do on filter buffers
            if an onset was found
                add it to the note beatmaps vector
```

Both tempo estimation and onset detection methods (**aubio_tempo_do** and **aubio_onset_do**) use a phase vocoder to obtain FFT windows the size of their FFT window size parameter for analysing the frequency spectrum (spectral content) of the audio signal (see Audio Signal Analysis (2.1.2)). The process happens every frame, illustrated in Figure 10.

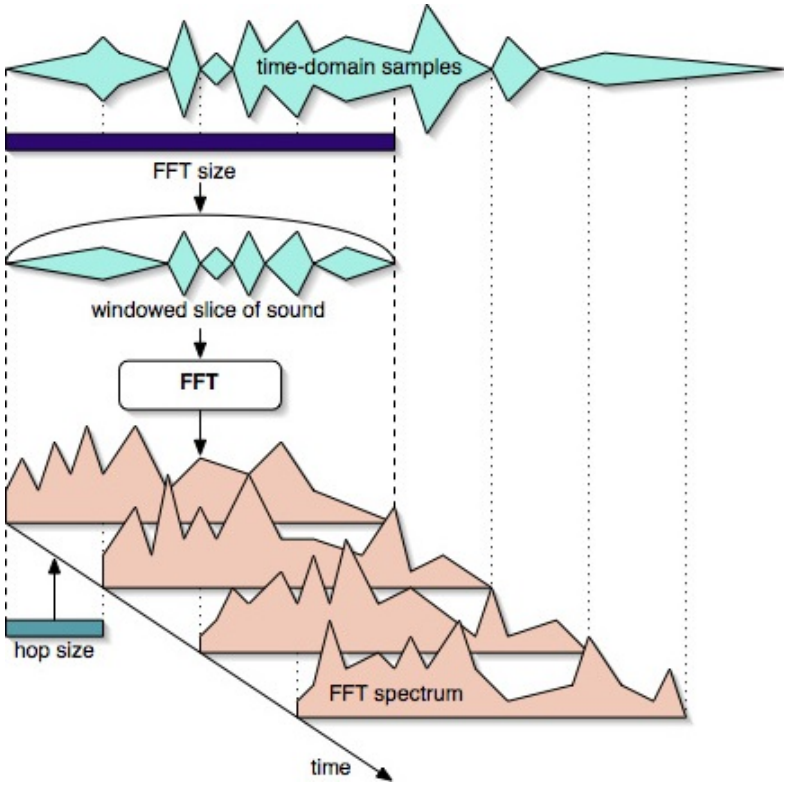


Figure 10: *Phase Vocoder with Overlap of 4 (Dudas and Lippe, 2006)*

The FFT window size must be higher than the hop size so that no samples are missed. The combination of hop size and window size affects the results of the next stages. An amount of overlap of the FFT windows can be defined as $overlap = \frac{windowsize}{hopsize}$. Overlap can be described as the number of FFT windows that each sample will be processed by, excluding the first few hops as seen in Figure 10. Hop size and window size must be powers of 2 so the most commonly

used overlap values are 2 and 4. Overlap of 1 is not ideal as the produced FFT windows don't form a complete description of the frequency domain. This is because FFT windows are usually tapered at the boundaries due to the use of a windowing function to reduce spectral leakage. A major problem with phase vocoding is the issue of resolution. Lower hop size increases time resolution while higher window size increases frequency resolution at the cost of "blurring" transients together, making time localization of onsets more difficult. Brossier (2007) uses overlap of 2 and 4 - or 50% and 75%, convertible to percentage using $overlap\% = 1 - \frac{100}{overlap}$ - when evaluating the onset detection functions implemented in *aubio*.

An **aubio_specdesc_t** object (short for spectral descriptor, encapsulates an onset detection function) is then used to reduce the signal using the detection function selected in the setup stage. Peak picking is performed on the reduced signal using a dynamic threshold based on weighting the median and the mean calculated from a window around the current frame around the user selected threshold to label onsets (Brossier 2007). The minimum time lag between onsets is equal to whichever is greater between hop size and the minimum inter onset interval. At this point, onset detection is completed as peaks identified as onsets are then appended to the beatmap note queues.

3.1.3 Tempo Estimation

Tempo estimation continues by performing beat tracking. Beat tracking uses an autocorrelation function (ACF) to identify beats from onsets by measuring the lag between onsets within a 6 second window. A bank of comb filter is then used to filter the ACF results into tempo candidates. The filter with the most energy corresponds to the lag of the ACF function - which is inversely related to the tempo such that the beats-per-minute (BPM) can be calculated as $BPM = 60 \cdot lag_{ms}$. The slower the BPM, the less probability it is given. The ACF is also biased towards longer lags, thus preferring slower BPM. These conditions result in estimates around a particular BPM being preferred - this value starts at 120BPM and changes as processing continues.

When approximately the same BPM has been detected three consecutive beats in a row, the algorithm enters a context-dependent mode where it considers previous beats to refine predictions of future beats. This allows smaller changes to be made to future beat predictions and BPM estimates. Confidence in the estimated BPM increases as more consecutive candidates are found to be similar. The algorithm simultaneously continues the initial mode of estimation so that if

a candidate differs greatly from the context-dependent mode, it can attempt to re-evaluate the continuity of BPMs more generally as it did in the beginning. The advantage of this two-mode system is that it can make small changes using the context-dependent mode while allowing for abrupt large changes using the initial mode.

This is a simplified description of the tempo estimation system implemented by *aubio*, described in full by Brossier (2007). The beat tracking and comb filter bank stages are visually similar to Figure 8c and 8d respectively.

In order to assist with selecting the correct tempo and offset value, a generating window is displayed while processing. Figure 11 shows this window.

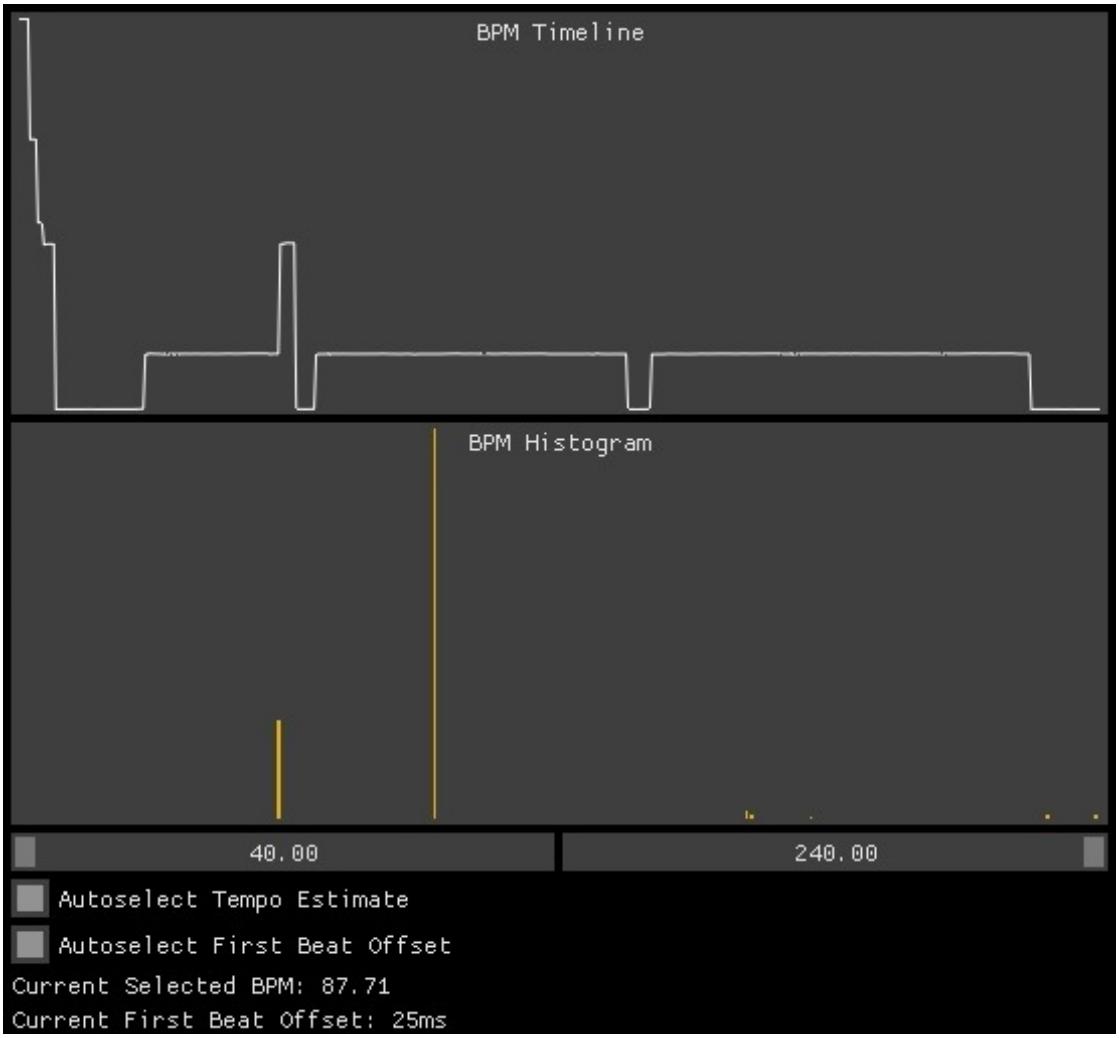


Figure 11: *Generating Window*

The generating window includes a timeline and a histogram of all BPM estimates. The timeline shows all estimates from the beginning of the song (left) to the current time (right). Below the timeline is a histogram of BPMs sorted

into 200 bins ranging from 40BPM to 240BPM. Values for the timeline and histogram are viewable by hovering with the mouse cursor (not shown). Bins with more estimates will peak higher therefore suggesting that bins BPM as an estimate. The histogram can be zoomed using the sliders below it, increasing the resolution of bins. The resolution of each bin is $\frac{max-min}{200}$ which makes the default resolution without zooming $\frac{240-40}{200} = 1BPM$. An option to use the highest confidence BPM selected by *aubio* is given as it is not necessarily equal to one of the BPMs suggested by histogram peaks.

In addition to picking the BPM, the offset of the first theoretical beat, B_0 must be picked. B_0 is "theoretical" as it need not correspond to a beat present in the music, it simply signifies when beats can start being placed using a beat interval, $B_{\Delta t}$. An option to auto-select the offset is available. This option will search the beats vector to find the beat with the closest BPM to the selected BPM, calculate the beat interval $B_{\Delta t} = \frac{60}{BPM}$ then calculate the timing of the first beat, B_0, by iteratively subtracting $B_{\Delta t}$ until $B_0 < 0$ then finally adding $B_{\Delta t}$ so that $B_0 > 0$.

It is important to note that this method of tempo estimation is only viable for songs that have a single tempo throughout. Variable tempo estimation requires structural segmentation of the music into sections where the tempo differs, which is not performed by the current method. An experienced user may be able to pick out tempos for several sections using the BPM timeline and histogram but no facility was created for adding several tempo sections to beatmaps.

3.1.4 Onset Function Training

When performing onset detection on songs that are not silent at the beginning, the onset detection functions do not have any previous FFT window data to compare with. This causes greatly increased sensitivity to detection in the beginning of the song, usually producing a large number of false detections. To combat this an option to train onset functions for a number of hops is provided. This processes the specified number of hops (default 200) but does not record the output for detections. After training is completed, the source buffer is reset back to the beginning of the song to begin processing normally with trained onset functions.

3.1.5 Onset Function Filtering

One of the available generation modes for beatmap generation developed uses filters to split the source buffer up into multiple filter buffers with filtered signals

for onset detection. This was done to detect onsets in different frequency ranges to explore the hypotheses of using filtering as a basic form of instrument separation and note categorization. Non-filtered mode is disadvantaged by the fact that it cannot detect notes occurring simultaneously whereas filtered mode can theoretically pick up as many simultaneous notes as there are filters - if instruments were separated perfectly. Games may also want to synchronize with notes within a particular frequency range, e.g. bass notes.

Initially, filtering was attempted using the an **aubio_filterbank_t** object but this object reduces FFT windows to a single energy value for each filter rather than sub-bands of the signal. Instead of using this object, the library *DSPFilters* (2012) was added to access signal filtering functionality. All filters used are 2nd order Butterworth filters. This filter type was selected empirically using *DSPFilters* accompanying executable to find a filter type which could be used to separate a signal into several bands without a significant amount of overlap while minimizing the loss of content between neighbouring bands. Figure 12 shows an example 2nd order Butterworth band pass filter.

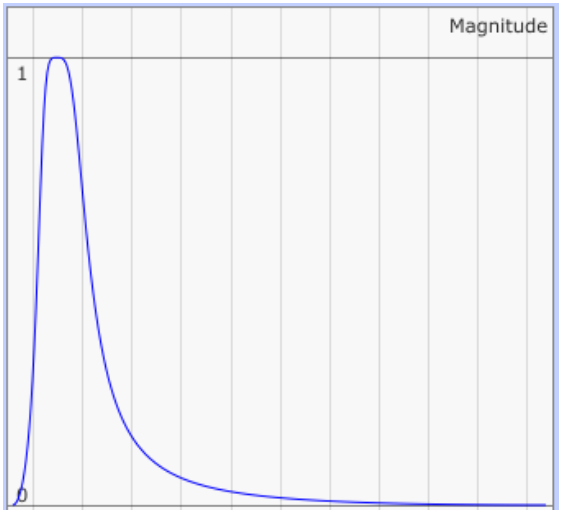


Figure 12: *Butterworth Band Pass Filter*

Currently there are 1-band, 4-band and 8-band filtering modes implemented. 1-band mode uses a single band pass filter where the centre frequency and width are user selected using two slider bars which appear on the generation settings window. In other modes, the first filter buffer contains the audio signal processed using a low pass filter while the last buffer contains the signal processed using a high pass filter. All of the buffers in between contain signals processed using band pass filters. The centre frequency and band width for each filter was empirically picked from the bands shown in Figure 5. Table 1 shows the parameters for each filter in both 4-band and 8-band modes.

For the 4-band mode, the bands correspond roughly to bass notes, low mid notes, upper mid notes and high notes. For 8-band mode, the bands correspond roughly to sub-bass, bass, upper bass, low mid, mid, upper mid, high notes and ultra high notes. The parameters for these modes were picked to be flexible and

Band	Type	Frequency	Width
1	Low Pass	300Hz	
2	Band Pass	500Hz	600Hz
3	Band Pass	1600Hz	1600Hz
4	High Pass	5000Hz	

Band	Type	Frequency	Width
1	Low Pass	42Hz	
2	Band Pass	100Hz	120Hz
3	Band Pass	230Hz	140Hz
4	Band Pass	500Hz	600Hz
5	Band Pass	1650Hz	1700Hz
6	Band Pass	3750Hz	3500Hz
7	Band Pass	7500Hz	5000Hz
8	High Pass	10000Hz	

Table 1: *Onset Detection Filters*

categorize notes broadly instead of attempting to pick out individual instruments from the frequency spectrum, so that the idea could be tested generically.

3.2 Filesystem

A filesystem was developed to enable storing beatmaps and songs used by *RhythMIR* between sessions. All files produced by *RhythMIR* are XML documents and have the extension ".RhythMIR". Boost filesystem (2016) is used to create directories, rename files and to move files to their directories while *RapidXML* (2009) is used to parse XML files when loading and saving to disk. Figure 13 shows the file structure for *RhythMIR*.

3.2.1 Song List

RhythMIR keeps track of songs using a song list file "_songs.RhythMIR" stored in the /songs/ directory. Since the directories that songs are stored in are based on the song artist and title, the song list only needs to store the artist, title and source for each song. The song list file structure is shown in Listing 2.

Listing 2: *Song List File Format*

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <songlist>
3     <song artist="artist" title="title" source="source.wav"/>
4     ... more songs
5 </songlist>
```

3.2.2 Beatmap List

Each songs directory has a beatmap list file "_beatmaps.RhythMIR" in their directory which lists the names of all beatmaps for the song.

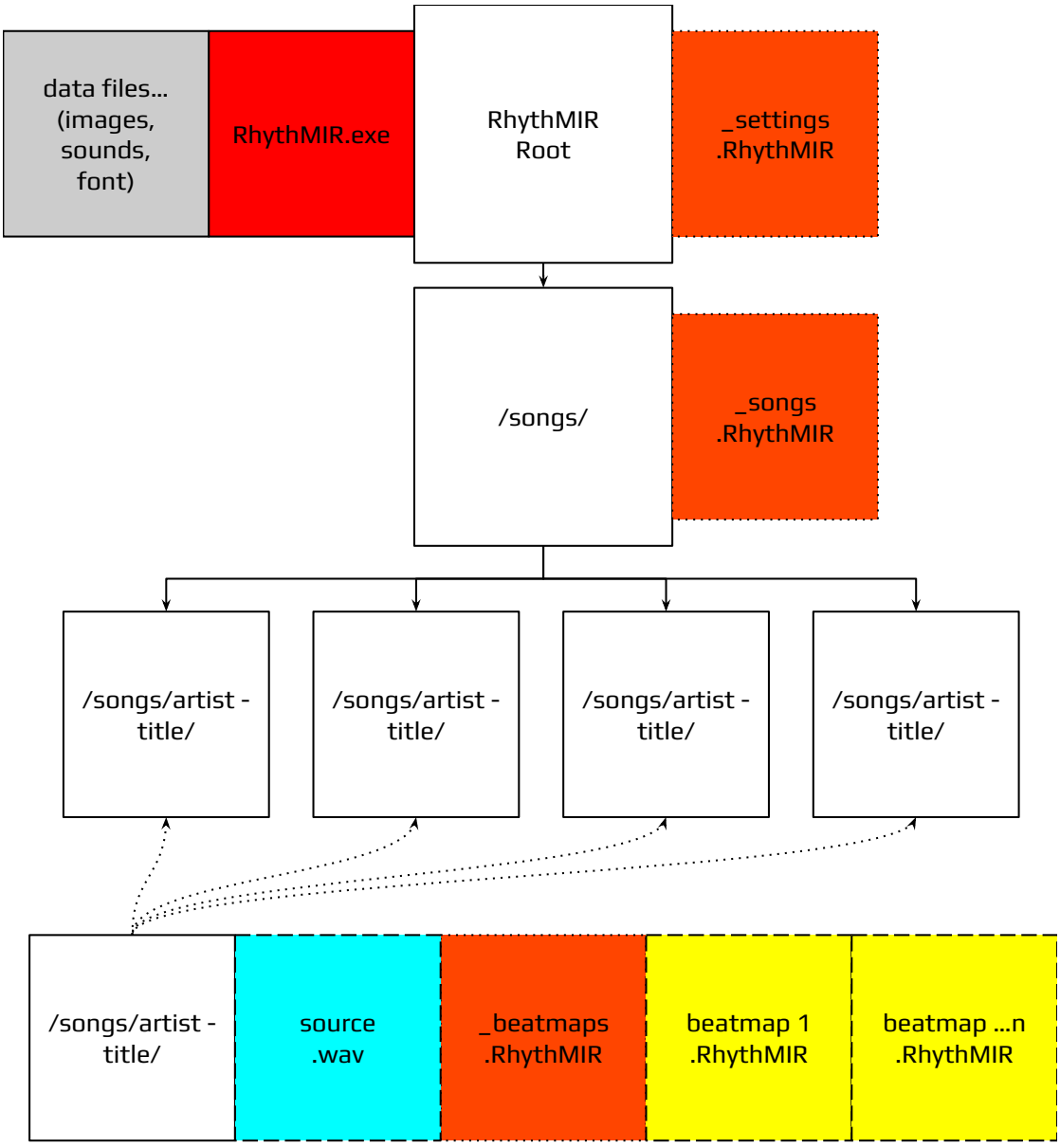


Figure 13: *File Structure*

Listing 3: *Beatmap List File Format*

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <beatmaplist>
3     <beatmap name="beatmap name"/>
4     ... more beatmaps
5 </beatmaplist>
```

3.2.3 Beatmaps

Each song can have any number of uniquely named beatmaps each stored in the file format in Listing 4.

Listing 4: *Beatmap File Format*

```
<?xml version="1.0" encoding="utf-8"?>
<beatmap artist="foo" title="bar" source="song.wav" type="4">
    <description>a description of foobar</description>
    <beats>
        <beat offset="1240"/>
        ... more beats
    </beats>
    <section BPM="200.000000" offset="20">
        <notequeue>
            <note offset="3118"/>
            ... more notes
        </notequeue>
        ... more notequeues
    </section>
    ... more sections
</beatmap>
```

Each beatmap can have any number of section nodes which correspond to timing sections with different BPMs within a song. Every beatmap currently produced only has one section as only single tempo songs are used. Each section has a number of note queue nodes which each stores a vector of onsets produced by beatmap generation as note nodes, e.g. a Four Key map will have four note queues. Optionally, if beats are being stored, a beats node will be present storing a number of beat nodes. The offset element for section, beat and note nodes indicates when a section, beat or node occurs within a song in milliseconds. Beatmaps are only partially loaded - only the artist, title, beatmap type and description - in the menu state to avoid unnecessary performance overhead when navigating beatmaps. Beatmaps are then fully loaded when transitioning from the menu state to the play state.

3.3 Application

RhythMIR has two states which implement the three major systems:

- **Menu State** implementing **beatmap generation** and the **filesystem**.
- **Game State** implementing the **rhythm game**.

3.3.1 Menu State

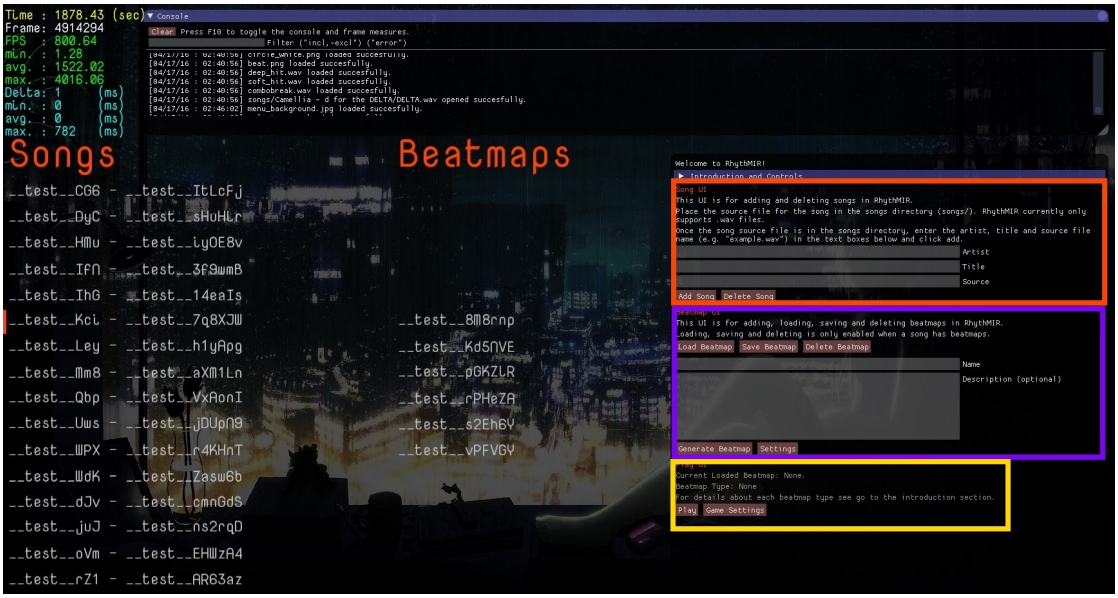


Figure 14: *RhythMIR Menu State*

Figure 14 shows an overview of the menu state.

Song/Beatmap Lists — Shown on the left in Figure 14. Displays available songs and their beatmaps. A selector shows what song or beatmap is currently selected. Navigating is done using WASD or the arrow keys. The selector moves between the song list and beatmap list.

Song UI — Outlined in orange on Figure 14, this UI contains buttons for adding and removing songs from RhythMIR. Every song must have an artist, a title and a source music file (only .wav files are supported for beatmap generation).

Beatmap UI — Outlined in purple on Figure 14, this UI contains buttons for generating new beatmaps, deleting beatmaps and opening the generation settings window. Each beatmap must be given a unique name and can optionally be given a description.

Play UI — Outlined in yellow on Figure 14, this UI shows the currently loaded beatmap, details about the beatmap, a button for changing to the play state and a button for opening the game settings window.

Console Window — Shown at the top of Figure 14. The console provides feedback for many actions in addition to notifying the user of any warnings or errors encountered. Pressing F10 toggles hiding the console.

In addition to what is displayed in Figure 14 above, there are three additional GUI windows for other purposes including the **Generation Settings Window** and the **Generating Window** covered in the Beatmap Generation section (3.1)). The third window is the game settings window with a number of widgets for changing the behaviour of the game.

3.3.2 Game State

The game state implements the **rhythm game**, developed in order to evaluate creating gameplay using MIR methods. The gameplay changes depending on what type of beatmap is being played and the selected game settings.

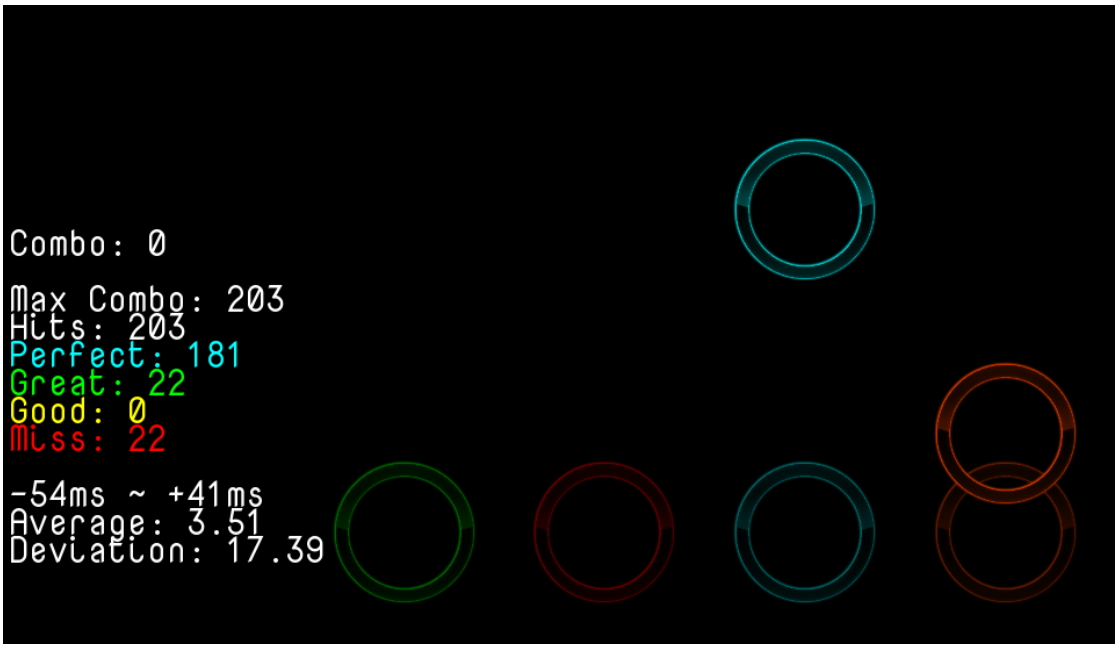


Figure 15: *RhythMIR Game State Zoomed In*

Figure 15 shows a four key beatmap being played. The game was designed similar to the classic arcade game *Dance Dance Revolution* (1998) with four lanes for notes to move along towards "receptors" or hit indicators which indicate when the player should hit a note. The game was designed this way as it is simple to implement while being similar to an existing rhythm game - which is important as the project aims to aid in content generation for existing games. If enabled, beat bars will also spawn based on the music BPM and offset and move towards the receptor area. Beat bars are not interactive but are useful to judge empirically if BPM and offset are correct. Single beatmap types can use the Shuffle (Table 2) setting to play as Four Key types.

Several performance statistics were implemented to assist in evaluating beatmaps, shown at the left in Figure 15. Perfect counts hits within $\pm 30ms$ from

the exact note offset, Great counts hits within $\pm 60ms$ and Good counts hits within $\pm 120ms$. Attempts within $\pm 300ms$ which do not fall in the other counters or where the circle goes off-screen are misses. Measures for the earliest hit, latest hit, average offset and standard deviation of notes hit are also calculated. These were implemented to help judge if notes in beatmaps are consistently well timed, which can be done empirically using the average hit offset and deviation.

Figure 16 shows the game settings window, with all available game settings described in Table 2 on the next page. Game settings modified from their defaults are saved to and loaded from the "**_settings.RhythMIR**" file between visits to the menu state.

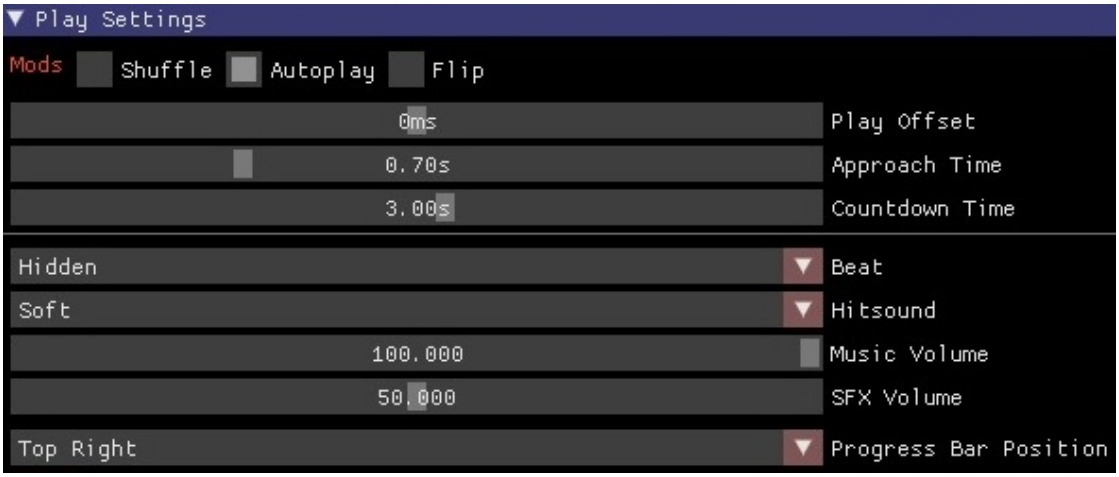


Figure 16: *Game Settings Window*

3.4 Windows, Graphics, GUI and Audio Playback

SFML (2015) was chosen for handling windows, events, 2D graphics and audio playback due to the previously developed extension library being available and ease of use. All resources used (textures, sound effects, music) in the project are loaded using *SFML* and cached in the global resource managers using their file names as keys until they are cleaned up either on exiting the current state that uses them or the application. The **sf::Font** class is used to load in the font used - NovaMono.ttf.

For creating GUI widgets, *dear imgui* (2016) was an obvious choice due to the ease of programming and the flexibility of control it gives. Adding new widgets such as buttons is simple, as shown by the example in Listing 5.

Setting	Description
Shuffle	Randomizes the path that each note spawns in.
Autoplay	Disables the note hit keybinds. The computer hits notes automatically when they reach the receptors.
Flip	Flips the play field, causing notes to spawn at the bottom and move towards receptors at the top.
Play Offset	Adjusts the offset that all notes are spawned at. Useful for testing beatmap timing (by playing the same map with different offsets) and fixing beatmaps that are off time without having to regenerate. Does not affect beats.
Approach Time	Changes the speed of notes/beats, measured in the amount of time to reach receptors after spawning.
Countdown Time	Amount of time at the beginning of playing to countdown before playing. Must be at least equal to approach time to allow the first notes to spawn.
Beat Type	Changes how beats are spawned. Available options are hidden, where no beats are shows, interpolated, where beat timings are calculated using the BPM and offset value of the song, and generated, where beats stored in the beatmap are used.
Hitsound	Changes the sound played when notes are successfully hit. Available options are none, soft and deep.
Music Volume	Changes the music volume.
SFX Volume	Changes the volume of all sound effects (hitsound and combobreak sound).
Progress Bar Position	Changes where the in-game progress bar is. Available options are top right, along top and along bottom.

Table 2: *Game Settings*

Listing 5: *Code for Button to open the Game Settings Window*

```
1 if (ImGui::Button("Game Settings"))
2     display_settings_window_ = !display_settings_window_;
```

When the button is pressed, it simply flips a boolean which then causes code elsewhere to toggle between rendering and not rendering the game settings window. *dear imgui* provides many functions for changing the layout of widgets such as **ImGui::SameLine** which places the next widget on the same line as the previous widget. The whole GUI is generated and sent for rendering every frame however, since the total number of vertices produced is low, the performance overhead is trivial.

3.5 Library Dependencies

A large number of software libraries were used to develop the systems in RhythMIR. Figure 17 shows an overview of dependencies. Briefly, these are:

Agnostic A personal C++ library implementing a number of utility classes and functions, e.g. the state machine and logger.

aubio (2015) A C library that provides the low level Music Information Retrieval functionality for the project encapsulated into several objects.

Boost (2016) A set of C++ libraries, RhythMIR uses the boost::filesystem library for manipulating directories and file paths.

dear imgui (2016) A C++ Immediate Mode Graphics User Interface (IMGUI) library used for creating all of the GUI widgets and windows in RhythMIR.

DSPFilters (2012) A C++ library of classes implementing a number of Digital Signal Processing (DSP) filters for manipulating audio signals.

RapidXML (2009) A C++ XML parser used for saving and loading the song list, beatmaps, beatmap lists and game settings.

SFML (2015) A C++ multimedia library used for main window management, user input, graphics rendering and audio playback.

SFML Extensions A personal C++ library of extensions to *SFML* including a rendering back-end for *dear imgui*.

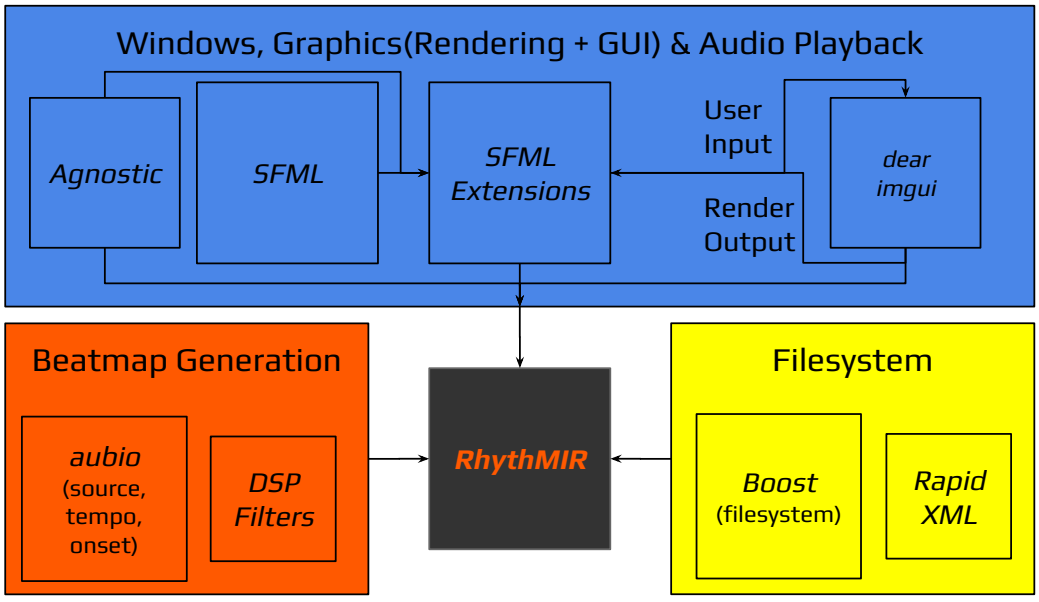


Figure 17: *Library Dependency Diagram*

4 Results and Discussion

- All music files were converted to .wav format with a samplerate of 44100Hz.
- All music files are complex mixtures across several genres of music since games generally include music of this type. The main genres included are Dance, Electronic and Rock since these are among the most common in rhythm games. A full list of all the songs used is available in Appendix A.
- **Hop Size** is the amount of samples or time to advance every frame of processing. Lower increases the time resolution of processing at the cost of increased computation time. The following hop sizes were available for testing:

$$\begin{array}{cccc} 16(<0ms) & 32(<0ms) & 64(1ms) & 128(2ms) \\ 256(5ms) & 512(11ms) & 1024(23ms) & 2048(46ms) \end{array}$$

- **Window Size** is the length of the FFT window used for obtaining frequency data, in samples. Higher increases the resolution of frequency data at the cost of increased computation time. The available window sizes is based on the selected hop size and available overlap values.
- **Overlap** is the amount of overlap between FFT windows. Overlap is calculated as $overlap = \frac{Windowsize}{Hopsize}$. Overlap of 2, 4 and 8 were made available for tests. Overlap of 1 caused anomalous results during testing (example shown in Figure 18). Overlap of higher than 8 caused a significant increase in computation time. Based on overlap, the following window sizes were made available for testing:

$$Hopsize \times 2(0-92ms) \quad Hopsize \times 4(1-185ms) \quad Hopsize \times 8(2-371ms)$$

4.1 Tempo Estimation

In order to evaluate the tempo estimation method, a selection of songs with known BPM and offset were collected. All songs used were obtained from and have beatmaps available on osu! (2007). This was done because these songs have already been through a timing process done by the beatmappers that created the beatmaps thus they have accurate BPM and offset values available.

To be considered useful for rhythm games (the strictest genre accuracy-wise) the generated BPM accuracy should be ± 0.1 of the reference value and offset of the first beat should be within $\pm 10ms$ of the reference value. Note that the reference offset will be the first beat in a song rather than the first beat of the first bar. This is because the developed system does not distinguish between beat types in

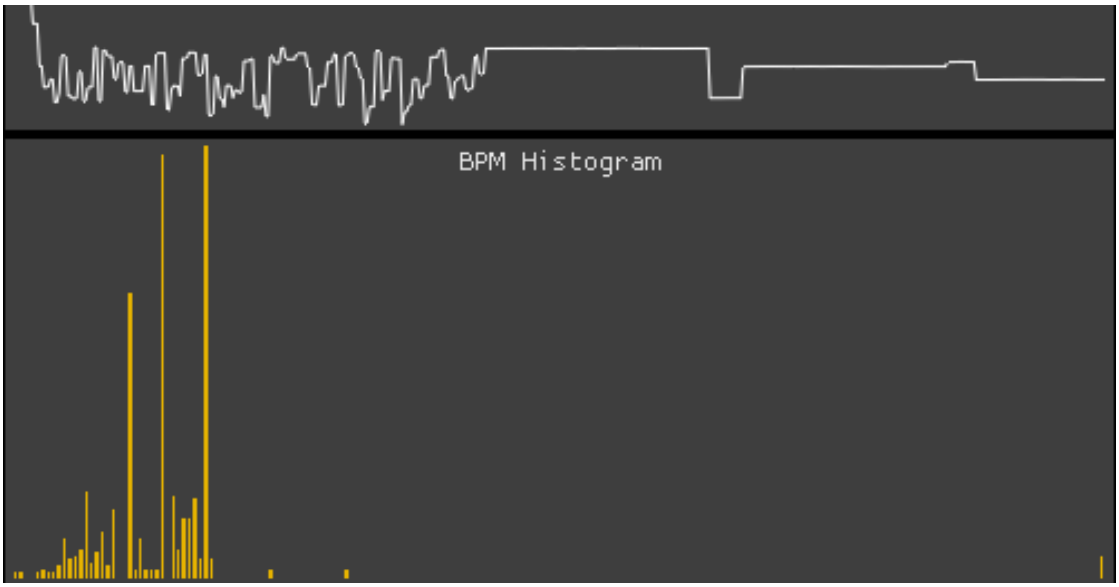


Figure 18: *Overlap 1 Anomaly - Tempo estimation method failing to find reasonable continuity between beats*

a bar. A beatmapper could easily increase the offset after generation by the beat interval to obtain the first beat in the first bar. Figure 19 shows 2 bars for music with 4 beats in a bar, labelling the beat types.

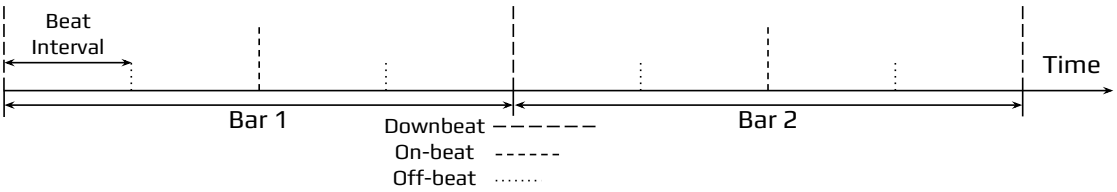


Figure 19: *Two Bars with 4 Beats in a Bar*

The amount of beats in a bar is defined by a time signature, e.g. $\frac{4}{4}$, where the upper number is the number of beats in a bar and the lower number is the note value for beats. The time signature is a high level concept used by musicians to define the relative duration of notes and beats. The tempo estimation method does not understand the structure of music - including time signatures or musical bars - it simply produces an estimate based on beats picked from onsets present in the music. Since the tempo estimation method prefers values around a particular BPM (default 120BPM), songs with a real BPM that greatly deviates from this value will have to be factored up or down to fix the detected BPM to the correct time signature. This will be done manually for the results below.

4.1.1 Parameter Selection

Firstly, the most effective set of parameters for the algorithm must be found. A small part of the data set put together was tested using different hop sizes (HS)

and window sizes (WS). Table 3 shows these results for two songs.

Song[1] HS/WS	BPM	β	$\check{\beta}$	Δ	Λ	$\check{\Lambda}$	Δ	Offset$_{ms}$	α_{ms}	Δ_{ms}
Odyssey[2]										
16 / 32	175	116.73	175.10	+0.10	116.73	175.10	+0.10	256_{ms}	335	+79
16 / 64	175	87.54	175.08	+0.08	116.74	175.11	+0.11	256_{ms}	416	+160
16 / 128	175	87.59	175.18	+0.18	116.74	175.11	+0.11	256_{ms}	453	+197
32 / 64	175	87.56	175.12	+0.12	116.83	175.25	+0.25	256_{ms}	438	+182
32 / 128	175	87.56	175.12	+0.12	116.82	175.23	+0.23	256_{ms}	439	+183
32 / 256	175	87.56	175.12	+0.12	116.82	175.23	+0.23	256_{ms}	440	+184
64 / 128	175	87.62	175.24	+0.24	116.94	175.41	+0.41	256_{ms}	623	+367
64 / 256	175	87.62	175.24	+0.24	116.93	175.40	+0.40	256_{ms}	624	+368
64 / 512	175	87.61	175.22	+0.22	116.91	175.37	+0.37	256_{ms}	610	+354
Cry Thunder[3]										
16 / 32	130	130.19	130.19	+0.19	130.03	130.03	+0.03	388_{ms}	190	-198
16 / 64	130	130.17	130.17	+0.17	130.03	130.03	+0.03	388_{ms}	160	-228
16 / 128	130	130.91	130.91	+0.91	130.16	130.16	+0.16	388_{ms}	29	-359
32 / 64	130	130.16	130.16	+0.16	130.06	130.06	+0.06	388_{ms}	312	-76
32 / 128	130	130.15	130.15	+0.15	130.34	130.34	+0.34	388_{ms}	310	-78
32 / 256	130	130.25	130.25	+0.25	128.79	128.79	-0.21	388_{ms}	322	-66
64 / 128	130	130.41	130.41	+0.41	130.57	130.57	+0.57	388_{ms}	402	+14
64 / 256	130	130.35	130.35	+0.35	130.52	130.52	+0.52	388_{ms}	336	-52
64 / 512	130	130.32	175.22	+0.22	130.42	130.42	+0.42	388_{ms}	304	-84

Table 3: *Tempo Estimation Parameter Results*
β denotes the auto-selected BPM.
$\check{\beta}$ denotes the BPM rectified to the correct time signature.
Λ denotes the BPM selected using the histogram.
$\check{\Lambda}$ denotes the BPM selected using the histogram rectified to the correct time signature.
Δ denotes the difference between the reference BPM and detected BPM.

[1]Song details are available in Appendix A
[2]Akira Complex - Odyssey (Au5 Remix)
[3]Dragonforce - Cry Thunder

Table 3 has three highlighted columns which are the auto selected BPM, the BPM selected empirically using the implemented histogram and the auto selected offset. In all of these columns, values closer to 0 are better. A number of observations can be made based on the results shown in addition to what was learned while noting results:

- Lower hop size produced a more accurate result in both songs.
- At 16 and 32 hop size, overlap of 4 can be clearly seen to be the most accurate. At 64 hop size, higher overlap can be seen to increase accuracy. However this is irrelevant as the lower hop sizes produced better results.

- The histogram often displayed high peaks at several BPMs hinting that confidence in the automatically selected BPM is low or the time signature is wrong. However the best way to verify correct time signature is to check empirically by playing the beatmap rather than using the histogram.

- The method of offset detection appears to be fundamentally flawed. This can potentially be explained as being due to an erroneous assumption when designing the offset selection. The *aubio* beat tracker selects beats from any onsets detected in the music at constant intervals, the implication of this is that what the beat tracker labels as a beat is not necessarily aligned with one of the beats in a bar. Detected beats could have been labelled from onsets at any point in a bar which can't be assumed to be one of the whole beats in a bar. The accuracy of offset detection would appear to be better if the beat interval used to iterate back to the first beat were subdivided allowing iteration back in smaller intervals (1/2 beats, 1/3 beats, 1/4 beats, etc.) but this would require knowing the time signature of the music in advance so that whole beats could be subdivided using the correct factor. Given these problems, further testing of the offset selection method was abandoned.

Table 4 on the next page shows results for more songs. Further testing only includes results for hop size of 16 and 32 both with overlap of 4.

From the results in Table 4 it can be concluded that the preferred parameters for tempo estimation are hop size 16 and window size 64. Except for Epiphany, the most accurate BPM for each song was generated using these preferred parameters. With a bit of knowledge of how the system operates, the BPMs picked from the histogram proved more accurate than the auto selected BPM. On several occasions there was no clearly defined histogram peak as shown in Figure 20. In such cases, or when several peaks were very close, the furthest left (lowest BPM) peak was chosen. The far left peak was not chosen in Figure 20 as it was not reasonable to assume it as a better candidate than the marked one, since it is noticeably shorter. In all cases where there is no obvious peak, picking the highest far left peak gave the best results.

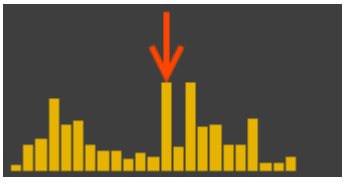


Figure 20: *Histogram Peak Picking*

Song[1] HS/WS	BPM	β	$\check{\beta}$	Δ	Λ	$\check{\Lambda}$	Δ
Odyssey[2]							
16 / 64	175	87.54	175.08	+0.08	116.74	175.11	+0.11
32 / 128	175	87.56	175.12	+0.12	116.82	175.23	+0.23
Cry Thunder[3]							
16 / 64	130	130.17	130.17	+0.17	130.03	130.03	+0.03
32 / 128	130	130.15	130.15	+0.15	130.34	130.34	+0.34
AugoEidEs[4]							
16 / 64	207	103.68	207.36	+0.36	103.57	207.14	+0.14
32 / 128	207	103.26	206.52	-0.48	103.62	207.24	+0.24
Flower Flag[5]							
16 / 64	170	113.37	170.06	+0.06	113.42	170.13	+0.13
32 / 128	170	113.45	170.18	+0.18	113.56	170.34	+0.34
Genryuu Kaiko[6]							
16 / 64	173	173.15	173.15	+0.15	115.39	173.09	+0.09
32 / 128	173	115.46	173.19	+0.19	115.5	173.25	+0.25
Everlasting Message[7]							
16 / 64	230	115.06	230.12	+0.12	115.10	230.20	+0.20
32 / 128	230	115.14	230.28	+0.28	115.18	230.36	+0.36
Epiphany[8]							
16 / 64	175	87.66	175.32	+0.32	87.54	175.08	+0.08
32 / 128	175	87.52	175.04	+0.04	116.80	175.20	+0.20
Dopamine[9]							
16 / 64	200	100.04	200.08	+0.08	100.05	200.10	+0.10
32 / 128	200	100.09	200.18	+0.18	100.09	200.18	+0.18
Average							
16 / 64				+0.1675			+0.11
32 / 128				+0.2025			+0.2675

Table 4: *Tempo Estimation Parameter Results*
β denotes the auto-selected BPM.
$\check{\beta}$ denotes the BPM rectified to the correct time signature.
Λ denotes the BPM selected using the histogram.
$\check{\Lambda}$ denotes the BPM selected using the histogram rectified to the correct time signature.
Δ denotes the difference between the reference BPM and detected BPM.

[1]Song details are available in Appendix A
[2]Akira Complex - Odyssey (Au5 Remix)
[3]Dragonforce - Cry Thunder
[4]DystopiaGround - AugoEidEs
[5]FELT - Flower Flag (MZC Echos the Spring Liquid Mix)
[6]Halozy - Genryuu Kaiko
[7]penoreri - Everlasting Message
[8]TwoThirds & Feint - Epiphany (feat. Veela)
[9]U1 overground - Dopamine

In several cases the auto selected BPM picked a more accurate BPM at some point throughout generation before then changing to a less accurate BPM while the histogram could be used to pick a more accurate BPM. This is because the auto selected BPM is selected from the longest chain of similar beat predictions,

which is not necessarily consistent with the most frequent tempo estimate.

4.1.2 Aggregate Results

The settled hop size and window size are 16 and 64 respectively. Table 5 shows results for many songs using these parameters. The %Δ column is introduced as it is suspected that results could be scaled by a percentage to increase overall accuracy. Only the most accurate result for each song is shown with yellow cells showing results picked from the histogram rather than auto-selected. Cells that had to be rectified by 1.5× are highlighted red and by 2× cyan.

Song	BPM	β	$\check{\beta}$	Δ	%Δ
Akira Complex - Odyssey (Au5 Remix)	175	87.54	175.08	+0.08	+0.04
Camellia - δ:for the DELTA	174	116.05	174.08	+0.08	+0.04
Camellia & DJ Genki - Feelin Sky	174	87.07	174.14	+0.14	+0.08
Dragonforce - Cry Thunder	130	130.03	130.03	+0.03	+0.02
DystopiaGround - AugoEidEs	207	103.57	207.14	+0.14	+0.06
FELT - Flower Flag (MZC Echos the Spring Liquid Mix)	170	113.37	170.06	+0.06	+0.04
Halozy - Genryuu Kaiko	173	115.39	173.09	+0.09	+0.05
penoreri - Everlasting Message	230	115.06	230.12	+0.12	+0.05
TwoThirds & Feint - Epiphany (feat. Veela)	175	87.54	175.08	+0.08	+0.05
U1 overground - Dopamine	200	100.04	200.08	+0.08	+0.04
UNDEAD CORPORATION - Everything will freeze	240	120.09	240.18	+0.18	+0.08
USAO - Chrono Diver -PENDULUMs-	184	92.03	184.06	+0.06	+0.03
xi - Blue Zenith	200	100.04	200.08	+0.08	+0.04
xi - FREEDOM DiVE	222	111.16	222.32	+0.32	+0.14
Average				+0.11	+0.054

Table 5: *Tempo Estimation Results*
β denotes the selected BPM.
$\check{\beta}$ denotes the BPM rectified to the correct time signature.
Δ denotes the difference between the reference BPM and detected BPM.

4.1.3 Discussion

- The average rectified BPM error is +0.054%. The average accuracy of detected BPMs can be immediately improved by offsetting detected values by this percentage.
- Out of 14 songs, 9 of the best results were picked automatically.
- Only 1 song did not require the detected BPM to be rectified before calculating BPM difference. As expected this songs BPM is close to the methods preferred BPM of 120.
- 9 songs had to be rectified by doubling their detected BPM.
- 4 songs had to be rectified by multiplying their detected BPM by 1.5.

With 9 out of 14 songs within ±0.1 of their reference BPM, the overall accuracy of detected BPMs is considered high enough to be used directly for rhythm games. Several of the other results would also be within ±0.1 after adjusting for the average % error. Given the amount of BPM rectifications done, it would be prudent to add a feature to the application which automatically displays several factors of the detected BPM as possible BPMs. Unfortunately, the method of offset selection was implemented naïvely due to the assuming that beats produced by the *aubio* tempo object could be used to estimate a first beat offset and produced unexpectedly poor results, rendering it essentially useless.

4.2 Note Generation using Onset Detection

Unlike Tempo Estimation, the results from Onset Detection cannot be directly compared with reference beatmaps from other games. This is because onset detection is used to perform the subjective task of generating notes to describe music rather than its original purpose of finding the location of all notes present in an audio signal. The reference beatmaps have notes annotated in each beatmappers individual style to describe the music the way they perceive it. However, a large amount of notes placed by beatmappers correspond to onsets present in the music as this creates gameplay that appeals to a large audience. Objectively correct onsets are generally considered well placed as they are less likely to be interpreted differently by different players.

Evaluation of onset detection and resulting gameplay will be two-fold. An objective analysis of onset detection functions will be performed to ascertain the best parameters for normal onset detection. Following this, case studies will be done for several songs. For each song, several beatmaps will be created using *RhythMIR* and subjectively evaluated. Each case study will involve beatmaps created from both single and 4-band filter mode, as these types are playable.

4.2.1 Objective Analysis and Parameter Selection

In addition to hop size, window size and resulting overlap, onset detection has 4 additional parameters. These are Peak-picking Threshold (PPT), Minimum Inter-Onset-Interval (Min-IOI), Silence Threshold and Delay Threshold. The version of *aubio* used implements 8 functions (parameters and functions described in Beatmap Generation Setup section (3.1.1)):

Energy	High Frequency Content (HFC)
Phase Deviation (PD)	Spectral Difference (SD)
Complex Domain (CD)	Kullback-Liebler (KL)
Modified Kullback-Liebler (MKL)	Spectral Flux (SF)

The objective accuracy of onset detection can be evaluated by comparing results directly with the source musical score of the song or annotations hand labelled objectively - as opposed to creatively or expressively as done in game beatmaps. For the songs used, the source musical scores are not available and individually annotating them would be unreliable, since this process is normally done by several experts and cross referenced. Fortunately, *aubio* has results noted by Brossier (2007), MIREX 2005 and MIREX 2006 for most of its onset detection functions. Since the large majority of music is expected to be complex mixtures (many onset types), only the results for complex mixtures are considered.

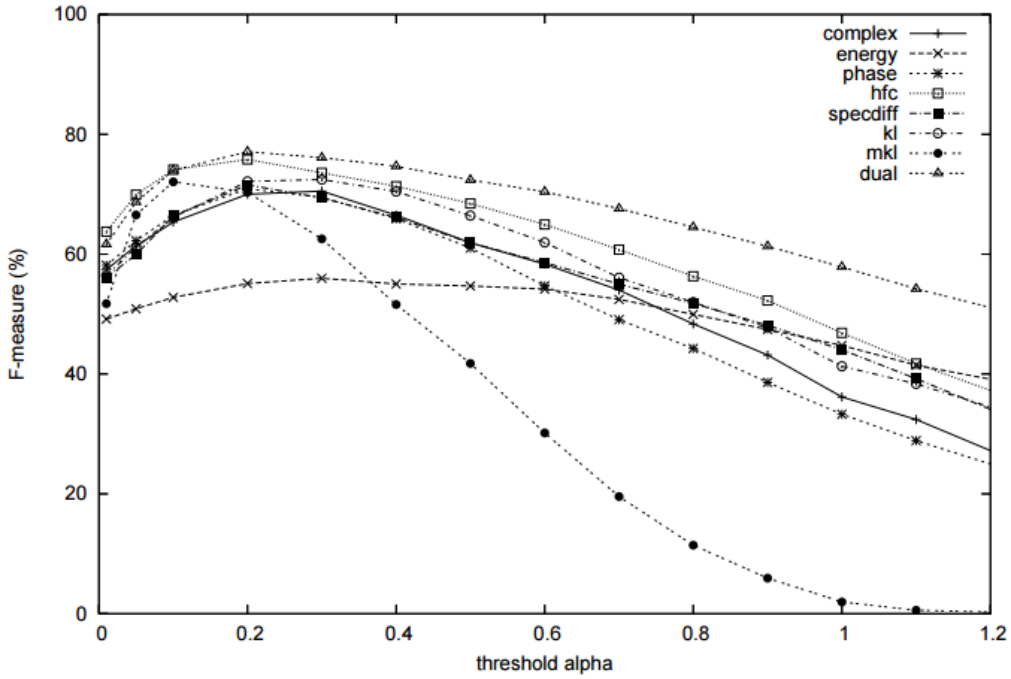


Figure 21: *aubio Complex Mixture Onset Detection Results (Brossier 2007)*

Figure 21 shows the F-Measure vs PPT for all functions except SF. The Energy function is instantly discarded due to its lack of consideration of the frequency content in a signal, resulting in poor results for complex mixtures. For all functions except MKL, a PPT of 0.2-0.3 appears to be optimal. The optimal PPT for MKL appears to be approximately 0.1. Spectral flux is similar in nature to spectral difference so its optimal PPT will be assumed to be similar at 0.2.

Figure 22 shows the table of MIREX 2006 results - with total, merged onsets and doubled onsets columns removed as they are not directly relevant. The dual function is also removed as it is not available for use.
Parameters is the PPT used for the function, the best performing PPTs are selected automatically. The precision column notes the amount of relevant onsets detected divided by total detections. The recall column notes the amount of

Contestant	Parameters	Avg. Correct	Avg. FP	Avg. FN	Avg. Precision	Avg. Recall	Avg. F-Measure
brossier_complex	0.3	503.8	227.2	208	0.689	0.695	0.682
brossier_hfc	0.2	577.2	275.8	134.6	0.670	0.786	0.713
brossier_specdiff	0.3	496.2	265.8	215.6	0.630	0.683	0.642

Figure 22: *MIREX 2006 aubio Complex Mixture Onset Detection Results*

relevant onsets detected divided by the total amount that should have been returned.

- CD returns the most reliable results, showing the highest precision value, but returns less of the total relevant onsets from the music.
- HFC appears to perform superior to the complex function, returning far more relevant onsets with only slightly less precision.
- SD appears to be inferior to CD and HFC.

Hop size implicitly defines the time interval between onsets being detectable, since only 1 onset can be labelled per hop. Higher window sizes increases the resolution of frequency content but reduces the accuracy of time localization. A balance between hop size low enough to distinguish onsets and window size high enough for good frequency resolution, while keeping time resolution reasonable, is therefore desirable.

Brossier (2007) examines results from MIREX 2005 and shows that a hop size of 128 (2ms time resolution) and window size of 512 (11ms time resolution, 43Hz frequency resolution) using a delay threshold of $4 \times Hopsize$ gave the best results for detecting onsets correctly and labelling them accurately for all functions.

All functions will therefore begin testing using these parameters. Attempts will be made to improve the output for each function until it is deemed that the function is ineffective or it is decided that reasonably optimal results are found.

4.2.2 Evaluation Metrics

Beatmaps are considered good if a large number of the onsets are subjectively agreeable with what is present in the music and the onsets can be considered reasonably well timed. In order to evaluate beatmaps concisely, four subjective metrics are defined:

Fidelity: How agreeable the notes generated are in the context of the music. Good fidelity means the majority of notes are relevant to the music.

Density: How many notes are present in the music relative to what was expected. Good density means few sections with unexpectedly high or low amounts of notes.

Detail: How finely detailed the notes placed describe the music. High detail means several notes were placed on what would be considered minor details. Note that Detail becomes irrelevant when Fidelity or Density is poor because it is difficult to determine the level of Detail described when there is not a reasonable number of relevant notes. As such, when Fidelity or Density is poor, Detail is not recorded.

Precision: How accurately notes are placed relative to where they were expected in time. High precision means the notes were positioned closer to where they were expected in time.

In addition to using these metrics, specific comments will be made for each function.

4.2.3 Case Study: FELT - Flower Flag (MZC Echoes the Spring Liquid Mix)

Flower flag is a relatively calm song with a consistent bassline, vocals and varying accompaniments. This song was chosen as the first case study as it is an example of the type of song that beatmap generation is expected to work well on. Table 6 shows the evaluated metrics for the song using all functions.

Function	Fidelity	Density	Detail	Precision
Single Function				
HFC	Very Good	Good	Very High	High
PD	Poor	Very Poor	n/a	Low
SD	Poor	Poor	n/a	Low
CD	Good	Average	High	High
KL	Good	Good	High	High
MKL	Average	Good	Average	High
SF	Very Good	Good	High	Low
4-Band Filtered				
HFC	Very Poor	Good	n/a	Very Low
PD	n/a	n/a	n/a	n/a
SD	n/a	n/a	n/a	n/a
CD	Good	Good	High	Low
KL	Average	Good	High	Low
MKL	n/a	n/a	n/a	n/a
SF	Very Poor	Poor	n/a	Very Low

Table 6: *Flower Flag Onset Detection Results*

HFC: Tested with 0.2 and 0.3 PPT with varying results.

+ Many finer details of the music were annotated when using 0.2 PPT and excluded when using 0.3 PPT. This suggests that 0.2 PPT could be used to produce harder difficulty beatmaps.
– Notes were noticeably off-time for one short section of the music.
– A few double detections were present. However, increasing the Min-IOI would easily remove these.

Phase Deviation: Tested with 0.1, 0.2 and 0.3 PPT and then 64 hop size and 256 window size with 0.2 PPT. Variation of parameters did not meaningfully change results.

+ At a few short sections the function placed notes particularly well. These sections contained a particular instrument producing pitched non-percussive sounds.
– For the majority of the song generated gameplay was very poor.

Spectral Difference: Tested 0.3 and 0.5 PPT with 0.5 producing better results.

– Notes were often noticeably out of time.
– Generated notes were sometimes agreeable but overall the results were very inconsistent.

Complex Domain: Tested with 0.2 and 0.3 PPT with varying results.

+ Similar to HFC, finer details were unveiled when putting the PPT down to 0.2 rather than 0.3.
– A substantial number of notes were slightly out of time.

Kullback Liebler: Tested with 0.3 and 0.5 PPT with 0.5 producing better results.

+ Overall note placement is good. Good amount of notes, consistently in time and agreeable.
– A few short bursts of far too many notes were present. Changing to 0.5 PPT minimized these bursts without compromising quality elsewhere.

Modified Kullback Liebler: Tested with 0.1 and 0.2 PPT with 0.1 producing better results.

– Notes were placed less reliably than the other Complex Domain functions (CD & KL).

Spectral Flux: Tested with 0.3 and 0.6 and 0.7 PPT with 0.6 producing the best results.

+ Often placed notes in places that other functions missed. Produces more notes overall than other functions. As such, the PPT was increased to reduce note count to a reasonable level.
± A number of notes were placed in arguably subjective locations, i.e. many people may disagree with them.
– Quite a lot of double detections present. Increasing Min-IOI could potentially remove them.
– A substantial number of notes were noticeably out of time.

4-Band Filtered Functions: For functions where beatmaps could be successfully generated, 4-Band mode functions produced similar gameplay. The only large variation is the Spectral Flux function which produced more notes than the others. Testing 4-Band mode for functions required each functions PPT to be increased dramatically to reduce the overall amount of notes generated. A default PPT of 0.8 was selected as the initial hypotheses for 4-Band functions. PPTs that produced the best results were:

HFC 0.8 **CD** 0.6 **KL** 1.0 **SF** 1.2.

– A large number of double/triple detections occur in all 4-Band beatmaps therefore the use of Min-IOI when creating 4-Band beatmaps is vital. A reasonable way to determine a good Min-IOI is to calculate the amount of time for a 1/4 notes based off of the songs BPM, e.g. for Flower Flag:

$$Min\text{-}IOI_{ms} = \frac{60 \cdot 1000}{170BPM \cdot 4} = 88ms$$

– The accuracy of 4-Band mode proved to be very low across the board.
– Many of the functions proved unsuitable for use with filtering as no reasonable beatmaps could be produced. These functions are marked with n/a on Table 6.

All single mode functions will be evaluated a second time in the next case study. Most functions will be excluded from future 4-Band results as they are believed to be ineffective, only CD and KL will be continued forward. While intelligible Four Key beatmaps could be generated using HFC and SF, the gameplay quality of the generated beatmaps was far too low.

4.2.4 Case Study: U1 overground - Dopamine

Dopamine is a chaotic song with a consistent but varying bassline, electronic vocals and many non-percussive transients. Dopamine was chosen as the second case study as it is the other extreme when compared to Flower Flag - onset detection is expected to work poorly on this song except for percussive notes. Table 7 shows the metrical results for Dopamine using the remaining functions.

Function	Fidelity	Density	Detail	Precision
Single Function				
HFC	Average	Average	High	Low
PD	Poor	Average	n/a	Low
SD	Very Poor	Poor	n/a	Low
CD	Good	Good	High	High
KL	Good	Good	Very High	High
MKL	Good	Very Good	Very High	High
SF	Good	Good	High	High
4-Band Filtered				
CD	Poor	Poor	n/a	Extremely Low
KL	Poor	Poor	n/a	Extremely Low

Table 7: *Dopamine Onset Detection Results*

HFC: Tested with 0.3 and 0.5 PPT with 0.5 producing better results.

± Many minor details were annotated, as a result many notes were placed in subjective locations, i.e. many people may disagree with them.
– Unlike before, the PPT had to be increased to reduce the amount of notes present. When doing so, many notes were still annotated questionably while many agreeable notes were removed.
– Notes were slightly off-time for most of the song.
– Similar to before, several double detections were generated.

Phase Deviation: Tested with 0.0 and 0.3 PPT with 0.0 producing better results.

+ Performed better than on Flower Flag, with a larger number of agreeable notes.
– Overall performance was still very poor.

Spectral Difference: Tested 0.3 and 0.5 PPT with 0.3 producing better results.

– Similar to Flower Flag, generated gameplay was very poor.

Complex Domain: Tested with 0.2 and 0.3 PPT with varying results.

+ Finer details were annotated when putting the PPT down to 0.2 rather than 0.3.
± Similar to HFC, many details were annotated with notes placed in subjective locations.
– A number of notes were slightly out of time.
– A large number of double detections were present.

Kullback Liebler: Tested with 0.3 and 0.4 PPT with 0.4 producing better results.

+ Similar to before, results are very good overall.
– Similar to before, a few short bursts of far too many notes were present. Changing to 0.4 PPT minimized these bursts without compromising quality elsewhere.

Modified Kullback Liebler: Tested with 0.1 and 0.0 PPT with 0.1 producing better results.

+ Produced very good overall results, better than on Flower Flag.

Spectral Flux: Tested with 0.5 and 0.6 PPT with 0.6 producing better results. Spectral flux performed very similarly to it's performance on Flower Flag compared to other functions.

4-Band Filtered Functions: Tested with PPTs: **CD** 0.6 **KL** 1.0; Min-IOI of 62ms was used to prevent excessive detections.

– Neither function tested produced a reasonable beatmap.
– The accuracy of 4-Band mode was exceptionally poor for Dopamine.

4.2.5 Discussion

- The metrics used to evaluate beatmaps are relative. For example, very good overall metrics for HFC on Flower Flag does not mean that the beatmap would be considered high enough quality for use as a beatmap in rhythm games. It simply means that output can be considered very good compared to other generated beatmaps and high quality enough to be used to aid in the beatmapping process.

- The best beatmap produced for each song is high enough quality to be used as a starting point by beatmappers in the beatmap creation process for rhythm games. An implication of this is that notes annotated by onset

detection are reliable enough to be used as suggested note locations for any games seeking to synchronize game elements with music.

- The best beatmap for Flower Flag was produced using the HFC function with 0.2 or 0.3 peak picking threshold.
- The best beatmap for Dopamine was produced using the MKL function with 0.1 peak picking threshold.
- The best beatmap produced for both songs was created using hop size 128 and window size 512.
- For both songs, a large amount of events in the music were not recognized as notes. The large majority of these were continuous sounds (in the signal they appear as steady state transients) which cannot be reduced by an onset detection function since they are not indicated by onsets, so they cannot be localized to a single point in time. Many games have gameplay objects that can be placed to describe these sounds, e.g. sliders in *osu!* (2007) or freeze arrows in *Dance Dance Revolution* (1998). As such, the beatmap generation process can be improved by exploring methods of annotating these sounds.
- Recommendations for what functions to use for Single mode generation on different types of music can be inferred from the case studies. The functions are listed below sorted by order of recommendation:
  - **SF** performed the best on average. This function is recommended as a good default choice.
  - **HFC** can be recommended for music where percussion is present and as a good choice in general. HFC produced average results on Dopamine, a song which is considered a poor match for the function.
  - **KL** performed the second best on average considering that it produced good results on both songs.
  - **MKL** performed only average on Flower Flag but very good on Dopamine. The behaviour of MKL is different from all other functions evaluated. As such it is recommended when the above functions fail to produce good results.
  - **CD** under-performed the above functions in general. This function is therefore not recommended.

  - **PD** is overall a very poor function applied to complex mixtures, however it can be used as an attempt to detect pitched non-percussive notes where other functions cannot.
  - **SD** is the worst performing function overall. It is not recommended for game content generation.

- 4-Band filtering mode could be used to produce reasonable beatmaps for Flower Flag but produced very poor beatmaps for Dopamine. Only CD and KL can be recommended when attempting to generate Four Key beatmaps.

- From the results of 4-Band filtering it seems that filtering is an ineffective way to categorize notes or obtain note frequency information. Filtering the source signal into several reduced signals had unintended side effects on many of the onset functions, causing them to be less effective or completely unusable. Better results may be possible with different filters or tweaked parameters. An alternative method of attempting to obtain frequency information about notes is to perform pitch detection on detected onsets.

- The preferred hop size and window size for tempo estimation is different from onset detection. This was unforeseen while developing *RhythMIR* and as a result the same hop size must be used, which also limits the choice of window size. Decoupling tempo estimation from onset detection should be done regardless of this point anyway since tempo estimation only needs to be performed once per song while onset detection must be performed any time a beatmap is generated.

5 Conclusion

The aim of the project was to explore how Music Information Retrieval (MIR) techniques could be used to assist in content creation for music video games. An application was developed which can perform four tasks via MIR:

- Tempo estimation with an average error of +0.11 BPM on the tested data set. This accuracy can be improved by offsetting results using average percentage error of +0.054%.
- First beat offset detection which proved to be unsuccessful.
- Note generation using onset detection to suggest positions to place game elements. Beatmaps generated using note generation were subjectively evaluated as good enough to assist in game content creation.
- Note generation using filtering to attempt detection of onsets occurring concurrently in a song and to categorize the generated notes by frequency. Filtering proved to be ineffective overall.

The developed system shows how Music Information Retrieval techniques can be used effectively to assist in content creation for music video games. Results produced by tempo estimation and non-filtered note generation are considered good enough to be used in the content generation process of rhythm games - the strictest genre of music video games quality-wise.

5.1 Future Work

Potential improvements to the developed system can be split into two main categories: direct improvements to the application and techniques to explore for improving content generation.

5.1.1 Application Features

Several improvements can be made to the developed application to ease the beatmap generation process:

- Note density graphs could be implemented to judge if the number of notes output by generation is reasonable before testing.
- Tempo estimation and onset detection are currently coupled while their optimal parameters differ. Tempo estimation only needs to be performed once per song so it should be performed separately.

- Having to manually check beatmaps for quality after generation took up a large portion of time while testing results.
- Since tempo estimation should be performed prior to beatmap generation anyway, a real time edit mode could be implemented where note suggestions are generated and selected by the user in real time. This would also allow changing detection functions on the fly so that the optimal function can be used for every section of a song.

5.1.2 Content Generation

Results could likely be improved by using superior performing functions such as the Neural Network function discussed in section 2.2.3.

The content generation system could potentially be improved by researching more advanced Music Information Retrieval techniques. Onset detection is one of the most basic forms of determining where to place notes. Notes produced by onset detections are limited to being recognized from onsets present in music. As a result, sounds that do produce distinct onsets such, e.g. non-percussive sounds in general, are mostly undetected. Note transcription methods that track event duration and pitch can potentially be used to detect these sounds. Pitch detection could also be used to categorize notes by their frequency rather than the filtering method attempted.

A different approach to note generation could take advantage of the subjective annotations done by human beatmappers. The neural network function discussed in section 2.2.3 could potentially be trained on beatmaps from rhythm games such as *osu!* (2007) rather than objectively correct data sets to learn the beatmapping styles of individual beatmappers.

References

Arentz, W. 2001. Beat Extraction From Digital Music. Available from: http://www.ux.uis.no/norsig/norsig2001/Papers/44.Beat_Extract_31820017132.ps [Accessed 10th March 2016].

aubio. 2015. (Version 0.4.2). [software library]. Brossier, P.M. Available from: http://aubio.org/ [Accessed 22nd February 2016].

Audiosurf. 2008. [computer game]. Windows PC. Fitterer, D.

Beat Hazard. 2011. [computer game]. Windows PC, PlayStation 3, iOS. Cold Beam Games.

Beatmania. 1997. [computer game]. Arcade. Konami. G.M.D.

Bello, J.P. and Sandler, M.B. 2003. Phase-based note onset detection for music signals. In: *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP '03)*. Hong-Kong. April 6-10. pp.441-444. [online]. Available from: doi: 10.1109/ICASSP.2003.1200001 [Accessed 19th April 2016].

Bello, J.P. et al. 2004. On the Use of Phase and Energy for Musical Onset Detection in the Complex Domain. In: *IEEE Signal Processing Letters*. 11(6): pp.553-556. [online]. Available from: http://www.eecs.qmul.ac.uk/former/people/jbc/Documents/Bello-SPL-2004.pdf [Accessed 21st March 2016].

Bello, J.P. et al. 2005. A Tutorial on Onset Detection in Music Signals. In: *IEEE Transaction on Speech and Audio Processing*. 13(5): pp.1035-1047. [online]. Available from: http://www.nyu.edu/classes/bello/MIR_files/2005_BelloEtAl_IEEE_TSALP.pdf [Accessed 10th March 2016].

Benetos, E., Dixon, S., Giannoulis, D., Kirchhoff, H. and Klapuri, A. 2012. Automatic Music Transcription: Breaking the Glass Ceiling. In: *13th International Society For Music Information Retrieval (ISMIR) Conference*. Porto, Portugal. October 8-12. pp.379-384. [online]. Available from: http://ismir2012.ismir.net/event/papers/379_ISMIR_2012.pdf [Accessed 17th March 2016].

Böck, S., Arzt, A., Krebs, F. and Schedl, M. 2012. Online Real-Time Onset Detection with Recurrent Neural Networks. In: *Proceedings of the 15th International Conference on Digital Audio Effects (DAFx-12)*. York, UK. September 17-21. [online]. Available from: http://www.cp.jku.at/research/papers/Boeck_etal_DAFx_2012.pdf [Accessed 15th April 2016].

Böck, S., Krebs, F. and Schedl, M. 2012. Evaluating the Online Capabilities of Onset Detection Methods. In: *Proceedings of the 13th International Society For Music Information Retrieval (ISMIR) Conference*. Porto, Portugal. October 8-12. [online]. Available from: http://www.cp.jku.at/research/papers/Boeck_etal_ISMIR_2012.pdf [Accessed 18th March 2016].

Böck, S. and Widmer, G. 2013. Maximum Filter Vibrato Supression for Onset Detection. In: *Proceedings of the 16th International Conference on Digital Audio Effects (DAFx-13)*. Maynooth, Ireland. September 2-6. [online]. Available from: http://phenicx.upf.edu/system/files/publications/Boeck_DAFx-13.pdf [Accessed 14th April 2016].

Böck, S. Krebs, F. and Widmer, G. 2015. Accurate Tempo Estimation based on Recurrent Neural Networks and Resonating Comb Filters. In: *Proceedings of the 16th International Society For Music Information Retrieval (ISMIR) Conference*. Málaga, Spain. October 26-30. pp.625-629. [online]. Available from: http://ismir2015.uma.es/articles/196_Paper.pdf [Accessed 23rd March 2016].

Boost. 2016. (Version 1.6.0). [software library]. Available from: http://www.boost.org/ [Accessed 1st April 2016].

Brossier, P.M. 2007. *Automatic Annotation of Musical Audio for Interactive Applications*. PhD thesis. Centre for Digital Music, Queen Mary University of London.

Carter, J. 2003. *THE FREQUENCY SPECTRUM, INSTRUMENT RANGES, AND EQ TIPS*. [online image]. Available from: http://www.guitarbuilding.org/wp-content/uploads/2014/06/Instrument-Sound-EQ-Chart.pdf [Accessed 13th April 2016].

Crypt of the NecroDancer. 2015. [computer game]. Windows PC, Mac OS, Linux, PlayStation 4, PlayStation Vita. Brace Yourself Games.

Dance Dance Revolution. 1998. [computer game]. Arcade. Konami.

dear imgui. 2016. (Version 1.4.8). [software library]. Cornut, O. Available from: https://github.com/ocornut/imgui [Accessed 4th March 2016].

Dixon, S. 2006. Onset detection revisited. In: *Proceedings Of The 9th International Conference On Digital Audio Effects.* Montreal, Canada. September 18-20. pp.133–137. [online]. Available from: http://www.dafx.ca/proceedings/papers/p_133.pdf [Accessed 18th March 2016].

DSP Filters. 2012. (Commit 6f2c1e3 November 2012). [software library]. Falco, V. Available from: https://github.com/vinniefalco/DSPFilters [Accessed 12th April 2016].

Dudas, R and Lippe, C. 2006. *Diagram of the Short Term Fourier Transform (STFT).* [online image]. Available from: http://www.richarddudas.com/publications/2006-cycling-dudas-lippe-pvoc/ [Accessed 18th April 2016].

Duxbury, C., Bello, J.P., Davies, M. and Sandler, M. 2003. Complex domain onset detection for musical signals. In: *Proceedings of the Digital Audio Effects Conference (DAFx-03).* London, UK. September 8-11. pp.90-93. [online]. Available from: http://www.eecs.qmul.ac.uk/legacy/dafx03/proceedings/pdfs/dafx81.pdf [Accessed 19th April 2016].

Eyben, F., Böck, S., Schuller, B. 2010. Universal Onset Detection with Bidirectional Long Short-Term Memory Neural Networks. In: *Proceedings of the 11th International Society For Music Information Retrieval (ISMIR) Conference.* Utrecht, Netherlands. August 9-13. pp.589-594. [online]. Available from: http://ismir2010.ismir.net/proceedings/ismir2010-101.pdf [Accessed 15th April 2016].

Foote, J. 1999. Visualizing Music and Audio using Self-Similarity. In: *Proceedings Of The Seventh ACM International Conference On Multimedia (Part*

1). Orlando, Florida, USA. October 30-November 5. pp.77-80. [online]. Available from: http://www.musanim.com/wavalign/foote.pdf [Accessed 17th March 2016].

Foote, J. 2000. Automatic Audio Segmentation Using A Measure of Audio Novelty. In: *Proceedings of IEEE Int. Conf. Multimedia and Expo (ICME2000).* New York, NY. July 30-August 2. pp.452–455. [online]. Available from: https://www.fxpal.com/publications/automatic-audio-segmentation-using-a-measure-of-audio-novelty.pdf [Accessed 20th March 2016].

Foote, J. and Uchihashi, S. 2001. The beat spectrum: A new approach to rhythm analysis. In: *IEEE International Conference on Multimedia and Expo.* Tokyo, August 22-25. pp.881-884. [online]. Available from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.93.8485&rep=rep1&type=pdf [Accessed 10th March 2016].

Guitar Hero. 2005. [computer game]. PlayStation 2. Harmonix.

Hainsworth, S. and Macleod, M. 2003. Onset detection in music audio signals. In: *Proceedings of the International Computer Music Conference (ICMC).* Singapore. [online]. Available from: http://quod.lib.umich.edu/i/icmc/bbp2372.2003.047/1 [Accessed 19th April 2016].

Huron, D. 2001. What is a Musical Feature? Forte's Analysis of Brahms's Opus 51, No. 1, Revisited. *The Online Journal of the Society for Music Theory.* 7(4). [online]. Available from: http://www.mtosmt.org/issues/mto.01.7.4/mto.01.7.4.huron.html [Accessed 10th March 2016].

Masri, P. 1996. *Computer Modeling of Sound for Transformation and Synthesis of Musical Signals.* PhD thesis. University of Bristol, UK.

MIREX. 2005. *Audio Onset Detection Results.* [online]. Available from: http://www.music-ir.org/mirex/wiki/2005:Audio_Onset_Detection_Results [Accessed 30th April 2016].

MIREX. 2006. *Audio Onset Detection Results.* [online]. Available from:

http://www.music-ir.org/mirex/wiki/2006:
Audio_Onset_Detection_Results [Accessed 30th April 2016].

MIREX. 2015. *Audio Onset Detection Results per Class.* [online]. Available from: http://nema.lis.illinois.edu/nema_out/mirex2015/results/aod/resultsperclass.html [Accessed 14th April 2016].

MIREX. 2015. *Audio Tempo Extraction Results.* [online]. Available from: http://nema.lis.illinois.edu/nema_out/mirex2015/results/ate/summary.html [Accessed 23rd March 2016].

MIREX. 2016. [online]. Available from: http://www.music-ir.org/mirex/wiki/MIREX_HOME [Accessed 23rd March 2016].

SFML. 2015. (Version 2.3.2). [software library]. Gomila, L. Available from: http://www.sfml-dev.org/ [Accessed 21st February 2016].

Soundodger+. 2013. [computer game]. Windows PC. Studio Bean.

O'Keeffe, K. 2003. *Dancing Monkeys*. [online]. Available from: http://monket.net/dancing-monkeys/ [Accessed 10th March 2016].

osu!. 2007. [computer game]. Windows PC, Mac OS. Dean Herbert.

osu!wiki. 2016. *Beatmapping*. [online]. Available from: https://osu.ppy.sh/wiki/Beatmapping [Accessed 8th March 2016].

RapidXML. 2009. (Version 1.13). [software library]. Kalicinski, M. Available from: http://rapidxml.sourceforge.net/ [Accessed 31st March 2016].

Yane, U. 1998. BMS Format Specification. [online]. Available from: http://bm98.yaneu.com/bm98/bmsformat.html [Accessed 8th March 2016]

Zapata, J. and Gómez, E. 2011. Comparative Evaluation and Combination of Audio Tempo Estimation Approaches. In: *AES 42nd Conference on Semantic Audio*. Ilmenau, Germany. July 22-24. [online]. Available from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.

354.83&rep=rep1&type=pdf [Accessed 23rd March 2016].

Bibliography

Dixon, S. 2006. Simple Spectrum-Based Onset Detection. [online] Available from: http://www.music-ir.org/evaluation/MIREX/2006_abstracts/OD_dixon.pdf [Accessed 18th March 2016].

Hess, A. 2011. Beat Detection for Automated Music Transcription: An exploration of Onset Detection Algorithms. [online]. Available from: http://bingweb.binghamton.edu/~ahess2/Onset_Detection_Nov302011.pdf [Accessed 10th March 2016].

Scheirer, E.D. 1997. Tempo and beat analysis of acoustic musical signals. [online]. Available from: http://www.iro.umontreal.ca/~pift6080/H09/documents/papers/scheirer_jasa.pdf [Accessed 18th March 2016].

Appendices

A Music Files and Beatmaps Used

Explicit permission to use files was obtained from Akira Complex, Camellia, Feint and FELT. The purposes the music is used for all fall under fair use - since files are only used for research analysis and are not distributed or commercialized in any way - so music from other artists is also used. Further details can be found on each songs beatmap page (source) including the reference BPM for each song. To view the reference offset the beatmap must be opened in edit mode inside *osu!*.

Artist	Title	Beatmapper	Source
Akira Complex	Odyssey (Au5 Remix)	ProfessionalBox	https://osu.ppy.sh/b/754143
Camellia	δ:for the DELTA	Naitoshi	https://osu.ppy.sh/b/817623
Camellia & DJ Genki	Feelin Sky	eLy	https://osu.ppy.sh/b/709939
DragonForce	Cry Thunder	Jenny	https://osu.ppy.sh/b/704360
DystopiaGround	AugoEidEs	Raikozen	https://osu.ppy.sh/b/764082
FELT	Flower Flag (MZC Echoes the Spring Liquid Mix)	Frostmourne	https://osu.ppy.sh/b/169450
Halozy	Genryuu Kaiko	Hollow Wings	https://osu.ppy.sh/b/433005
penoreri	Everlasting Message	Azer	https://osu.ppy.sh/b/677938
TwoThirds & Feint	Epiphany (feat. Veela)	Aiceo	https://osu.ppy.sh/b/687303
U1 overground	Dopamine	fanzhen0019	https://osu.ppy.sh/b/494818
UNDEAD CORPORATION	Everything will freeze	Ekoro	https://osu.ppy.sh/b/555797
USAO	Chrono Diver -PENDULUMs-	Skystar	https://osu.ppy.sh/b/925161
xi	Blue Zenith	Asphyxia	https://osu.ppy.sh/b/658127
xi	FREEDOM DiVE	Nakagawa-Kanon	https://osu.ppy.sh/b/129891