SwarmPlayer: A Swarm-Based Application For Music Visualisation

Basil Regi
Student ID: 1527413
Supervised by: Achim Jung

April 10th, 2020

Submitted in conformity with the requirements for the degree of:
MSci Computer Science with Industrial Year
at the School of Computer Science, University of Birmingham

Abstract

The relationship between the auditory and visual experience of listening to music has been studied for a long time. Music visualisers were created as an exploration into that idea, using various different techniques. This paper explores how music visualisers are used and created, and also into the concept of swarm behaviour, to propose an interesting and novel approach to visualising music. All of the research is then combined in the production of the SwarmPlayer application; a prototype of the proposed swarm-based music visualisation technique.

Acknowledgements

I would like to thank my family and friends for their incredible support in the completion of this project. I would also like to thank Professor Achim Jung for his amazing supervision throughout the year and for the entertaining meetings which always ended with some new idea to explore.

Contents

1 Introduction

Music, at its very fundamental level, aims to encapsulate the listener and transfix them in a state that can provoke specific emotions and feelings within them. Listening to music for the most part is a single-sensory action, as majority of the time it is a background task. Frith describes music as 'the soundtrack to everyday life' in [14], as people now listen to music whilst carrying out their day-to-day activities, such as cooking, studying or commuting. However, there are certain scenarios in which listening to music is the primary action being undertaken, where the listeners are solely focused on the different characteristics that make up the piece of music they are listening to. They are free to feel the emotions conveyed by the music, be it at a night-club, a concert or even in the comfort of their own homes. This is where music visualisers play an important role, providing an immersive experience when listening to music, making it a multi-sensory activity.

1.1 Audio-Visual Links

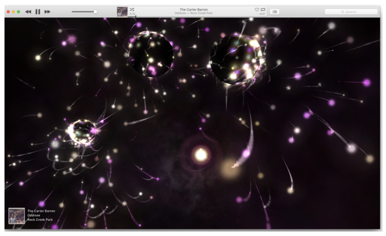
The correlation between auditory and visual perception in humans has been studied for a long time. Previous research within this field has revealed various ways in which music can influence the perception of visual scenes [5], a prime example of this would be the use of music in film. The use of music within films dates back to the origins of film itself, and very few people would deny that music contributes to their experience of the film [8]. This idea is further supported by a study conducted by Boltz [4], in which participants were presented with ambiguous film clips accompanied by positive, negative or no music and asked to make judgements based on the clip they viewed. The study found that, relative to the control group of no music, positive and negative music significantly biased viewers' interpretation of a film clip. However, this poses the question, does the inverse relationship hold true?

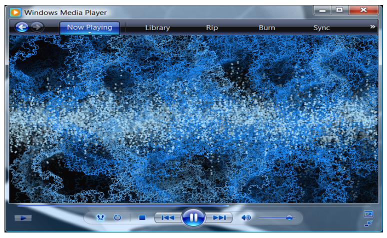
There is very little research into how visual information influences our perception of music [3]. However, in [5], Boltz, Ebendorf and Field carried out experiments to investigate precisely this relationship. They carried out two experiments; in Experiment 1, listeners were presented with ambiguous tunes paired alongside visual displays with varying emotion provoking imagery (positive and negative) or with no visual information at all (as a control group). Experiment 2 was conducted in a similar manner but instead of focusing on the characteristics of each tune, listeners were given an unexpected recognition memory task; where they were required to distinguish an old tune in its original form to a version that had been increased or decreased in either tempo, pitch or both. The results of the experiments consolidated the use of music visualisations as it proved that the visual displays influenced how the music was perceived by the listeners. Experiment 2 further extended this idea and showed that the use of visual displays distorted melody recognition in a mood congruent fashion. Consequently, this indicates that the use of an appropriate visualisation of music, which matches the mood intended to be conveyed by the composer, can supplement the listening experience and provide a deeper understanding of the music to listeners.

1.2 Examples of Music Visualisations

Music visualisation can be defined as the process of interpreting sound with some form of visual imagery [18]. More recently this term has been commonly associated with computer generated imagery provided as part of music playing applications such as iTunes (See 1a [22]) and Windows Media Player (See 1b [26]). Although these provide vivid imagery which loosely relates to the music, the field of music visualisation is a lot more varied.

Figure 1: Visualiser Examples


(a) iTunes Visualiser


(b) Windows Media Player Visualiser

There are two main categorisations of music visualisations; augmented score visualisations and performance visualisations [21]. Augmented score visualisations focus on the presentation of musical information to the viewer. These forms of visualisations produce graphics similar to that of the staff-like notation used to transcribe music [30], and although they convey a lot of information about the exact nature of the piece of music, they usually require a good background in music theory to be able to fully understand. In contrast to this, performance visualisations deal with different musical characteristics such as volume, mood, tempo, etc., that can be extracted from an audio stream [30]. They then aim to find a mapping between the audio characteristics on to a visual rendering of some sort. The visual renderings can range from simple frequency spectra, with a few frequency bands, to relatively elaborate animations that react to the different audio features [38]. These auditory-visual mappings are what make performance visualisations more intriguing than augmented score visualisations as they can still provide information about the underlying qualities of the piece of music in a more understandable manner while also providing visually aesthetic imagery to enhance the listening experience.

1.2.1 MidiVisualiser: Interactive Music Visualisation using VRML

MidiVisualiser[17] is a music visualisation framework created by Graves, Hand and Hugill at the Centre for Technology and the Arts, De Montfort University. It is an immersive virtual reality tool that aims to take a first step towards the creation of a virtual environment in which there is a direct correlation between what is heard and what is seen (See Figure 2). Unlike other visualisation tools, MidiVisualiser is also interactive, meaning it can be used as a compositional aid by allowing users to manipulate, create and remove musical features from the midi file within the tool.

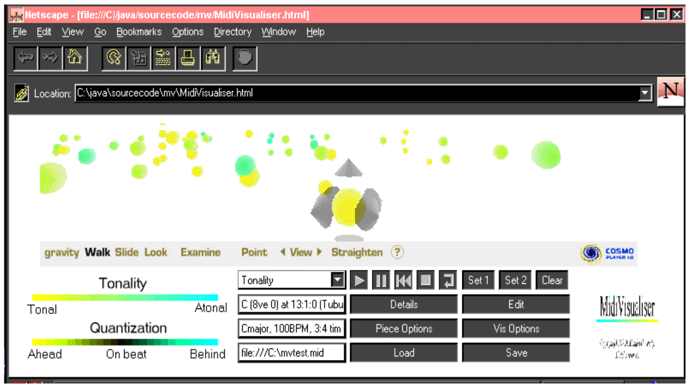

Netscape - [file:///C|/java/sourcecode/mv/MidiVisualiser.html]
File Edit View Go Bookmarks Options Directory Window Help
Location: C:\java\sourcecode\mv\MidiVisualiser.html
gravity Walk Slide Look Examine Point View Straighten
Tonality
Tonal
Atonal
Quantization
Ahead
On beat
Behind
Tonality
C (8ve 0) at 13:1:0 (Tubu
Cmajor, 100BPM, 3:4 tim
file:///C:\mvtest.mid
Set 1
Set 2
Clear
Details
Edit
Piece Options
Vis Options
Load
Save
MidiVisualiser

Figure 2: MidiVisualiser Java Applet and Embedded VRML World

MidiVisualiser falls under the augmented score visualisation category mentioned previously. Although the visualisation does not follow under the typical score notation, it adopts a 'piano-roll' like structure commonly used in modern day applications to represent the information extracted from the midi files. Typical 'piano-roll' based visualisations display note timings and pitch mapped to x and y coordinates (possibly even making use of colour), however, MidiVisualiser takes this one step further. The two types of information represented within MidiVisualiser are micro-level (characteristics such as pitch of a single note) and macro-level (structural characteristics such as density of notes in a passage or coherence of a run of notes). Figure 3 shows the arbitrary mapping of spacial dimensions to musical properties used by Graves, Hand and Hugill in the creation of MidiVisualiser.

$x - z$ dimension	$\leftrightarrow$	Beginning time of note
Length along $x - z$ dimension	$\leftrightarrow$	Length of note
Orientation in x dimension	$\leftrightarrow$	Key on velocity
Orientation in z dimension	$\leftrightarrow$	Key off velocity
y dimension	$\leftrightarrow$	Stereo placement
Colour	$\leftrightarrow$	Pitch
Shape	$\leftrightarrow$	Instrument
Size	$\leftrightarrow$	Volume

Figure 3: Spacial Dimensions to Musical Property Mappings

Although testing of MidiVisualiser was conducted on a small sample size, there was a wide range of musical experience within the participants. The participants were given a 30 second test excerpt of music and asked to view a static visualisation, then to view an animated visualisation with sound, and finally, they were given the ability to manipulate the structure of the excerpt.

The tests validated that the visualisation was coherent with the participants' natural interpretation of musical form. They were also able to correctly interpret the tonality mapping without ever having heard the piece, which further supports the effectiveness of the visualisation. One of the participants even reported that when viewing the animation, the separate experiences of seeing colours and hearing sounds began to fuse together; creating a 'synaesthetic' experience. This further reinforces the idea of how an effective visualisation can elevate the listening experience.

1.2.2 Sonic Visualiser

Sonic Visualiser[7] is characterised as a desktop application for the analysis, visualisation and annotation of music audio files. It was created by Cannam, Landone and Sandler at the Centre for Digital Music, Queen Mary University of London, mainly with the intention to provide a visualisation platform to be used by researchers within the centre. Sonic Visualiser contains a number of high-quality, general purpose visualisation tools such as waveform and spectrogram views which allow for musical analysis. It also allows for users to annotate the audio with features such as beat times. The Sonic Visualiser tool combined with automatic audio analysis tools, such as the Vamp plugin, forms a powerful tool to visualise and understand music audio files to a much higher level. Sonic Visualiser is an interesting example of visualisation as it falls into the performance visualisation category mentioned previously, however, instead of producing visually aesthetic renderings, it produces graph-like visualisations of the music to convey complex information about the characteristics of the music file. Figure 4 below shows an example of Sonic Visualiser in use, with a stereo music audio file loaded.

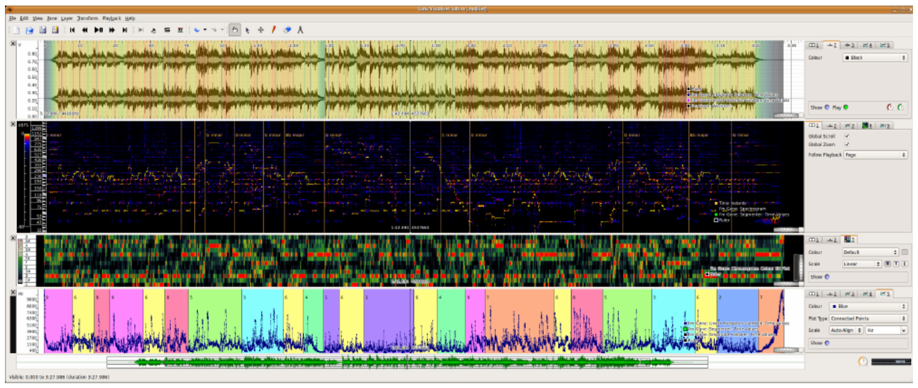


Figure 4: Sonic Visualiser

Figure 4 shows the pane and layer paradigm used by Sonic Visualiser to allow users to have multiple visualisations open at a time, all aligned by the time axis. Each visualisation pane itself can have multiple layers (overlaid data representations):

- The topmost pane shows the audio as a waveform, overlaid with the color gradient from a note onset detection likelihood function.
- The second, larger pane contains a spectrogram visualisation in peak-frequency estimation

mode. This is then overlaid with the output from a Vamp plugin which performs key estimation (depicted by a line and label for each estimated key change).

- The third pane contains the output from a Vamp plugin which calculates the pitch chroma distribution [1].
- The final pane contains two layers. One layer is a spectral centroid plot [1] and the other layer shows coloured segments based on timbre.

The visualisations displayed within the Sonic Visualiser tool, such as waveforms and spectrograms, are very common visualisations used to display audio files once they have been loaded into an application. These are very useful for conveying information to its viewers but aren't always visually aesthetic and requires some experience to be able to fully understand what is being displayed.

1.2.3 CymaSense: A Real-time 3D Cymatics-Based Sound Visualisation Tool

CymaSense [25] is a visualisation tool built by McGowan, Leplâtre and McGregor at Edinburgh Napier University. It is a music visualisation application inspired by the field of Cymatics, which is a term coined by Hans Jenny for the impressions created by sound on physical mediums such as water [23]. These impressions create interesting quasi-3D patterns which can be observed using a chladni plate. Cymatics provide an appealing choice for visualising music because not only does it produce interesting and mesmerising patterns which can encapsulate the viewers, it is also a concrete representation of the sound being played. Although it does not necessarily provide very informative visualisations, the patterns created can be mapped to different frequencies and therefore could prove as a great indicator of pitch within sound. CymaSense does exactly this by finding mappings between features within music files onto these cymatic patterns that are produced. The table below shows the various mappings that are used within CymaSense:

Feature	**Mapping to Cymatic Pattern**
Amplitude	Scale of Cymatic Pattern and the particle size
Pitch	Specific Cymatic Pattern/Shape
Pitch	Colour (Higher pitch = Lighter colour)
Sound Brightness	Surface Quality of Cymatic Shape (Darker sounds = More rounded shape)

Figure 5: CymaSense Mappings

With these mappings, CymaSense produces visualisations which pulsate whilst also changing in colour and shape according the sound being played at the time. Figure 6 shows a sample output

[1]These terms are specific features that can be extracted from audio files, it is not discussed in-depth as part of this project.

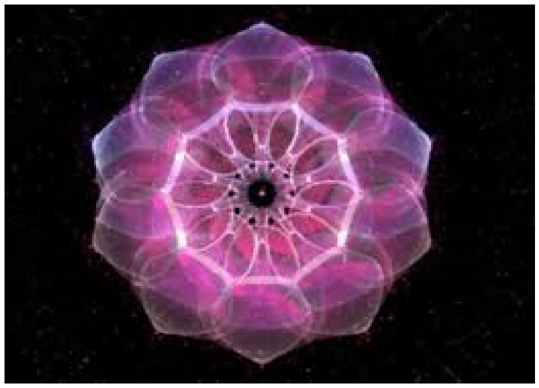


Figure 6: Sample Output from CymaSense

from the CymaSense tool [2]. CymaSense was implemented using Cycling 74' Max [9], which allows for real-time audio and midi processing, and Unity [39] for real-time 3D graphics generation. The tool also allows several options for the user to customize the visual output, such as being able to change the default Cymatic shape and particle size.

CymaSense is a good example of a performance visualisation tool as it is visually aesthetic whilst also conveying information about the different features within the music being played. It supports the use of naturally occurring phenomenons to create effective computer visualisations for music.

1.2.4 Dance: A Visualisation of Music?

All the examples explored so far are of computer generated imagery which are used to reflect the music being played in some automated way. However, an interesting example to think about is how the field of dance can be seen as visualisation of music. In fact, the Merriam-Webster dictionary defines the term 'music visualisation' as a dance that constitutes a direct translation of music into motion [29]. Dance as an art form has been around for a long time, and it is an interesting concept as it exemplifies the natural link, formed by humans, between music and the visualisation of it using motion. It has become embedded within our culture, with several different genres of dance being used to express the different genres of music in a wide variety of ways. Dancers and choreographers naturally do what this project is aiming to achieve using computers, which is to extract information from the music: such as tempo, rhythm/beats, timbre etc., and find interesting mappings to motion using groups of dancers in a group performance or different body parts in a solo performance.

When considering dance, the imagery that most often comes to mind is that of duets or of ones in small groups of less than 20/30. However when considering the idea of swarms, they can have upwards of 200 bodies participating within them, moving together in some coordinated manner. This idea can be equated to dance on a much larger scale, involving larger crowds of people. An example of this is a 'flash mob'; a term coined for a large public gathering at which people perform

[2]Visit `https://johnmcgowanphdwork.wordpress.com/about/` for more information about CymaSense and videos of example visualisations.

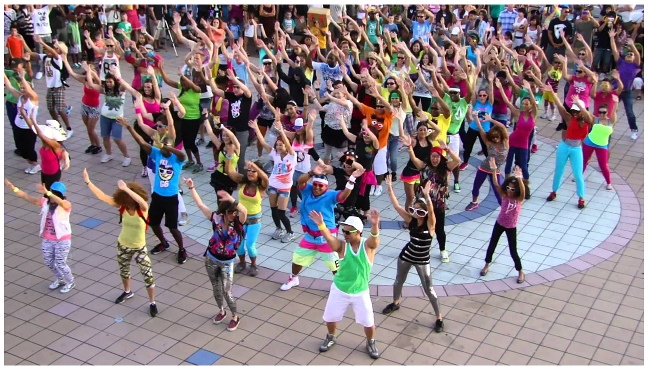


Figure 7: Example of a 'Flash Mob'

[12]. Although flash mobs aren't specifically about dance performances, majority of them are impromptu dances organised using social media. These performances, although choreographically quite simple, can be visually quite intricate and appealing (see Figure 7 [13]). They use the large number of people in the group to form complex patterns and designs with very simple and memorable choreography. This again poses an interesting link between dance and the use of swarms as a visualisation technique for music.

A more in-depth study into how dancers interpret music to produce such fascinating choreographies could help provide a more concrete mapping between musical features and visual displays. However, as part of this project, the focus will be on the idea of mapping musical features onto choreographed motion between large groups of people to produce interesting visualisations.

2 Swarm Behaviour

2.1 What is Swarm Behaviour?

Swarms are a naturally occurring phenomenon which has been observed in nature among collections of living organisms, but only recently have scientists started researching deeper into how this phenomenon occurs and what the possible applications of it could be. A swarm by definition is a large group of insects moving together [37], however, the idea of swarm behaviour extends further than just insects; it can be seen in schools of fishes swimming together [35], flocks of birds flying together, and even amongst humans in large crowds [20]. These swarms form quite intricate and mesmerising displays which are visually appealing. As discussed earlier, the movement and behaviour of these swarms can be equated to dance in a more spontaneous sense, rather than a fully choreographed piece. Visualisation of these swarm systems using simulations can provide an interesting idea for music visualisation.

Swarm simulation systems are usually multi-agent systems where the agents are able to interact with each other and their environment. The agents usually follow a very simple rule set, with no centralised control structure determining their behaviour. These simulations usually aim for realistic recreations of natural behaviours to allow study into the idea of swarms further. Although modelling accurate swarm behaviour can provide useful solutions to certain problems, this project will focus more on the visual appeal of the intricate structures and patterns created by the agents within these swarm systems.

2.2 Modelling Swarm Behaviour

Swarm behaviour modelling involves trying to find mathematical models to accurately summarise the behaviour of insects and animals in these swarm structures. By being able to successfully model these behaviours, a better understanding of the underlying physics in these decentralised systems can be gained, and possibly even aid the use of it within modern technology. One of the main uses of swarms is in the field of Robotics, where the aim is to design robust, scalable, and flexible collective behaviours for the coordination of large number of robots [6].

Over the years there have been various attempts to accurately model a swarm, but the fundamental principles behind these models are based on the system proposed by Reynolds [31] on the visualisation of flocking behaviour in birds. Reynolds states that for a bird to participate in a flock, it must have behaviours that allow it to coordinate with the other birds in the flock. He then goes on to say that these flocks consist of two balanced but opposing behaviours; a desire to stay close to the flock and a desire to avoid collisions within the flock [35]. The idea of these 'steering behaviours' are expanded on in a later paper by Reynolds [32] where an agent, in a multi-agent system, is presented using a simple vehicle model on which these behaviours act as a steering force to shift its direction of motion.

2.2.1 Vehicle Model

Reynolds presents the idea of modelling these autonomous agents as ‘vehicles’ in an abstract sense to encompass a wide range of transportation, from planes and cars to, albeit a stretch, even the legs of living organisms.

The paper also describes how the idea of these steering behaviours can be independent of the underlying locomotion scheme, which means that the specific embodiment of motion, i.e. speed and direction of wheels on a car, aren’t linked to the direction of steering. They simply implement the direction of motion as desired by the agent’s ‘brain’. Figure 8 shows the three-level hierarchy of motion behaviour proposed by Reynolds, which is similar to the model proposed by Blumberg and Galyean in [2].

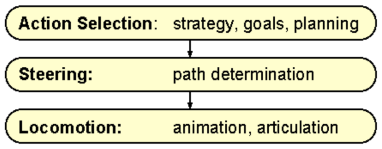

Action Selection: strategy, goals, planning
Steering: path determination
Locomotion: animation, articulation

Figure 8: Hierarchy of Motion Behaviours

The vehicle model is based on a point mass approximation, which is defined by a position and a mass. The model is also given a velocity attribute that is affected by various steering forces, which in reality, are limited in magnitude by factors in the environment. However, for the sake of simplicity, this limitation is summarised by a maximum force attribute. Most vehicles are also limited by a top speed, which is usually due to the acceleration and deceleration of various forces acting on the vehicle. Again, to simulate this in a realistic and simplistic way in the model, each vehicle is given a maximum speed attribute. This maximum speed is enforced by limiting the magnitude of the velocity vector by the scalar value of the maximum speed attribute.

The physics behind the model is based on forward Euler integration, which in simplistic terms mean that at each simulation step, the next state of the model is determined by the current state and the output of some function on the current state.

$$y_{n+1} = y_n + f(y_n)$$

At each time step of the simulation, a steering force is calculated using a desired direction determined by the steering behaviours and is applied to the point mass of the vehicle. The steering force is limited to the maximum force value and the speed of the vehicle is also limited to the maximum speed value. See algorithm below[3]:

```
vehicle {
    mass          scalar
    position      vector
    velocity      vector
    max_force     scalar
    max_speed     scalar
    }
```

```
steering_force = limit(mass(desired_velocity −
    velocity)/Δtime, max_force)
acceleration = steering_force / mass
velocity = limit(velocity + (acceleration ∗ Δtime),
    max_speed)
position = position + (velocity ∗ Δtime)
```

[3]For simulation purposes, mass and time can be ignored as mass is constant and Δtime = 1 at each time step.

As discussed earlier, this vehicle model was proposed as an abstraction to represent agents with motion, meaning it possesses the limitations that such a generic abstraction would. However, these limitations only affect the realism of the simulation. When considering the visual aesthetics, absolute realism is not required and therefore this model provides enough complexity to achieve the required swarming behaviour without needing to worry about the specifics of locomotion.

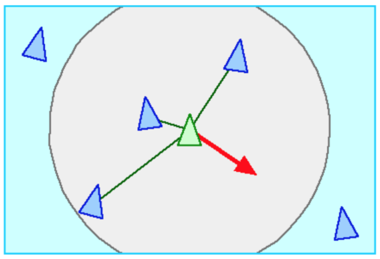
2.2.2 Steering Behaviours

The concept of steering behaviours is based on the assumption that locomotion is executed by a simple vehicle model as described above, and is summarised into one single steering force vector. Steering behaviours are therefore described as a vector representing a desired steering force. These steering behaviours allow implementation of motion behaviours studied in nature by simplifying them into vector quantities which affect the heading of the agent.

Reynolds proposes various behaviours in his paper [32] that can be categorised into two sections; individual behaviours and group behaviours. There are various individual behaviours such as seeking, fleeing and path following presented in the paper. These are characterised as individual behaviours as they are only concerned with the individual agent and a specific goal for them. It does not take into account anything regarding the agents' environment or other agents in the system. Group behaviours on the other hand are behaviours implemented by each agent as a reaction to other agents within their local neighbourhood. This local neighbourhood is determined by the simulated perception model given to every agent in the system. Although individual behaviours can aid a more realistic simulation, Reynolds outlines in [31] that only three group behaviours are required for flocking (a synonym for swarming in birds). Therefore this project will focus specifically on those behaviours.

Separation

Separation behaviour allows an agent to maintain a certain distance from other agents nearby. The steering force for separation is computed by retrieving the positions of all the neighbours within the agent's neighbourhood and then calculating a vector away from them (see Figure 10). The implementation of this behaviour is done by calculating the direction vector towards the agent in question from every agent in its neighbourhood, which is then scaled inversely to the distance between them. More simply, this means the closer the neighbouring agent is, the more the agent in question wants to move away from it. All the direction vectors are then summed up to produce a net direction in which the agent should move to avoid its neighbouring agents, which is then used as the desired velocity. See algorithm below:

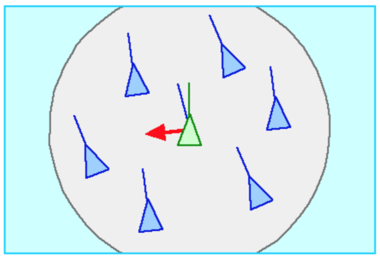


```
\\ neighbours is a list of agents in our agents
    neighbourhood
for each agent a in neighbours
{
     direction = position − a.position
     distance = length(direction)
     scaled_direction = normalise(direction) ∗
         (1 / distance)
     desired_velocity += scaled_direction
```

Figure 10: Separation

Alignment

Alignment behaviour allows an agent to align itself with its neighbours by adjusting its velocity to match that of its neighbours (see Figure 11). Implementation of this behaviour is simple; the average velocity of the neighbouring agents is used as the desired velocity. To find the average velocity; the velocities of all the neighbouring agents simply need to be summed and then divided by the number of agents within the neighbourhood. See algorithm below:

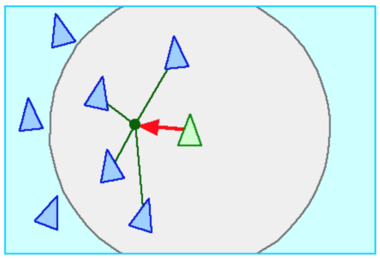


```
\\ desired_velocity is initialised as a zero−
    vector
for each agent a in neighbours
{
     desired_velocity += a.velocity
}
desired_velocity = desired_velocity /
    neighbours.size
```

Figure 11: Alignment

Cohesion

Cohesion behaviour is the behaviour that allows agents to approach other agents and form clusters with them (see Figure 12). Cohesion is achieved by calculating the average position of the neighbouring agents and then steering the agent towards that position. Implementation of this behaviour is simple in terms of the basic algorithm for it but can be quite complex depending on the world model being used in the simulation. See algorithm below:



```
\\ average_position is initialised as a zero-
    vector
for each agent a in neighbours
{
    average_position += a.position
}
average_position = average_position /
    neighbours.size
desired_velocity = average_position - position
```

Figure 12: Cohesion

2.2.3 Combining Behaviours

As stated earlier, the behaviours outlined above act as building blocks for more complex behavioural patterns such as 'flocking'. These behaviours need to be blended together in a balanced way to allow for the interesting group behaviours to emerge. However, balanced does not necessarily mean that all behaviours need to be considered at all times or even that they need equal weightings when being applied together. So, the combination of these behaviours is something that has to be considered carefully. Combining these behaviours can be done in two ways. One method involves agents switching between behaviours sequentially based on the situation and environmental factors. Specific events such as predators approaching can trigger the behavioural switch and goal change of the agent. These changes occur in the action selection stage of the behavioural model presented earlier and require complex rule sets to determine the correct behaviours to apply based on the situation.

Another method for combining behaviours is to blend them together so that every behaviour is factored in at each simulation step. The simplest way of achieving this is to compute each of the steering behaviour forces and then sum them together, possibly with a weighting factor for each of them. So to achieve flocking, the following equation would be used:

$$\overrightarrow{V} = c_1\overrightarrow{V_1} + c_2\overrightarrow{V_2} + c_3\overrightarrow{V_3}$$

where

$$\left\{\begin{array}{ll} c_1 \text{ to } c_3 & \text{are weightings for the corresponding vectors} \\ \overrightarrow{V_1} & \text{is the steering force vector from Separation behaviour} \\ \overrightarrow{V_2} & \text{is the steering force vector from Alignment behaviour} \\ \overrightarrow{V_3} & \text{is the steering force vector from Cohesion behaviour} \\ \overrightarrow{V} & \text{is the overall steering force vector for the agent} \end{array}\right\}$$

However, this method has two limitations which need to be considered. The first limitation is the computational complexity that is associated with this method, as there are a large number of calculations to be made at each simulation step, making this method computationally heavy. Another limitation is how the combination of these behaviours could mean that they cancel each other out in certain scenarios.

The weightings c_1 to c_3 will allow for control of the flock by manipulating what behaviours should have priority at each simulation step. Adjusting the weightings of these behaviours determine how close or far away each of the agents in the system will stay from each other. This means that dynamic and interesting clusters can be created within the swarm by affecting the weightings of each behaviour at each simulation step. This could allow the interactivity required to make the swarm simulation react to music.

In the paper, Reynolds also goes on to suggest a hybrid method of combining these behaviours, which is referred to as "prioritised dithering". This idea prioritises each of the behaviours and uses the behaviour with the highest priority which returns a non-zero value as the steering force at that simulation step. This eliminates the issue of opposing forces cancelling each other out and reduces the number of calculations. However, Reynolds goes on to state, after several re-implementations of the simulation, the simple linear combination of the behaviours proved to be sufficient.

3 Music Information Retrieval

Music Information Retrieval (MIR) is a field of study which focuses on the automatic extraction of musical information from various representations of music. It is an interdisciplinary field which encompasses areas such as Computer Science, Music Theory and Digital Signal Processing [16]. Due to the wide variety of ways that music is produced, represented and used [36], finding automatic techniques for information extraction which works across this broad spectrum proves to be a challenge within MIR. Although the use of automatic information retrieval to the field of music is an idea that has been around since the 1960's [24], it has gained considerable momentum with the recent growth of applications such as Spotify, that allow for instant streaming of musical material in digital form.

The increasing volume of digital music makes information retrieval a necessity, yet very few effective techniques exist to tackle this problem. Discovery of such techniques is a vast area of research in itself; one that is out of the scope of this project. Instead, as part of this project, the focus will be on finding an existing algorithm which is efficient and accurate enough to give the required information about any given piece of music, such that, an interesting visualisation can be created for it. Before delving into the various algorithms to do this, an exploration into the distinct forms in which music can be represented is required.

3.1 Music Representations

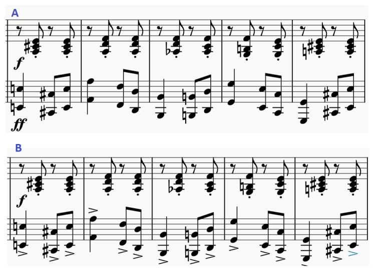
Music can be represented in a wide variety of forms depending on the information that it is trying to convey. Earlier, some examples of music visualisations were discussed, considering only the visual appeal but not how the music is being represented. Representations of music can be classified into three different groups: sheet music, symbolic and audio. Sheet music stands for the visual representations of a score in printed form or as digitised images, symbolic representations encapsulate any score representation with an explicit encoding of the notes or other musical events in the piece, and audio representations refer to the representations of the acoustic sound waves [28].

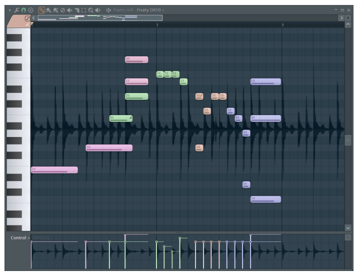
These representations each convey some information about the music, for example, sheet music can convey how the piece of music should be played live but requires a strong knowledge in music theory to be able to interpret. Whereas, an audio representation displayed as a graph can show the the basic structure of the music in an abstract sense and can be interpreted with little theoretical knowledge. There is no single unifying representation which can convey all the information possible, and different representations can suit different genres, i.e. symbolic representations can be good for electronic music genres such as EDM [4], however, isn't always complex enough to capture the intricacies of the way acoustic music is played.

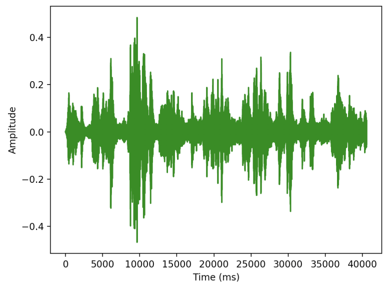
Although no one representation has any obvious advantage over the others, availability of music

[4]EDM stands for Electronic Dance Music which is a genre of music that heavily uses electronic and synthesised instrumentation.

in these representations is a factor that has to be considered as part of this project. Classical music can be quite easily found in sheet music representations and live performance covers of them can be found in audio format, however, very few symbolic representations of them can be found. Electronic music on the other hand can sometimes be found in symbolic representations, but is most commonly released in some audio format. Audio representations are one of the most common formats used in the modern day, which means it provides the most availability amongst the different genres of music. Furthermore, the use of audio format will also allow for the music to be played digitally behind a visualisation of it. Therefore, this project will specifically focus on MIR using the audio format.


(a) Sheet Music


(b) Symbolic



(c) Audio

Figure 13: Examples of Music Representations

Audio

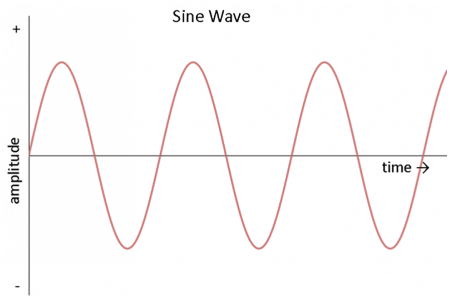


Figure 14: A Sine Wave

A brief understanding of how acoustic signals are represented digitally in audio is required to be able to to understand how information can be extracted from it. Sound waves travel through a medium by vibrating the particles in that medium back and forth at a particular frequency. A single note creates a sine wave like the one shown in Figure 14 [19]. However, more complex sounds produce a whole array of waves, which are summed together as it reaches our ear drum and converted into an electrical signal that is passed to our brain. Over time, humans have all learned how to differentiate these signals to be able to recognise different sounds. Computers deal with audio in a similar manner; a microphone (or other recording device) is used to convert acoustic/analogue signals into machine-readable electrical signals.

There are two crucial terms associated with a digital audio signal; the sample rate and the bit depth. The sample rate represents the frequency at which the input device records the amplitude value of the analogue signal being converted. This means that, in theory, the higher the sampling rate, the more accurate the audio representation is of the acoustic signal (better quality). The human ear is capable of hearing sounds with frequencies up to around 22kHz, which means that a sampling rate of around 44-48kHZ (the two main industry standards) would be sufficient enough to be able to capture the full cycle of the highest frequency wave audible by humans. The bit depth is the value associated with the range of values for the amplitude that can be recorded. So a 16-bit audio file would be able to represent 65,536 different levels of amplitude in a signal, this is further split into half to represent the positive and negative side of a signal. This is usually enough to be able to release a completed piece of music, however, when recording in raw signals, there can be such variation in the audio signal that it is difficult to accurately represent with only 16 bits. The industry standard recommendation is to record at 24-bit depth but audio is usually manipulated at 32-bit depth within computers to allow for the increase of amplitude due to signals being combined together.

3.2 Audio Feature Extraction

Audio feature extraction is the sub-field of MIR which focuses primarily on audio files. Audio features are contextual information that can be extracted from an audio signal [27]. The features extracted from the audio signals can be used to characterise them and provides abstract information about the sound that the signal represents. Usually, audio features are extracted at two levels; short-term frame level and long-term clip level [1]. A frame is the same as a single sample from the audio signal, but combines more than one channel if multiple channels exist, and a clip consists of a sequence of frames. Figure 15 exemplifies the relationship between frames and clips.

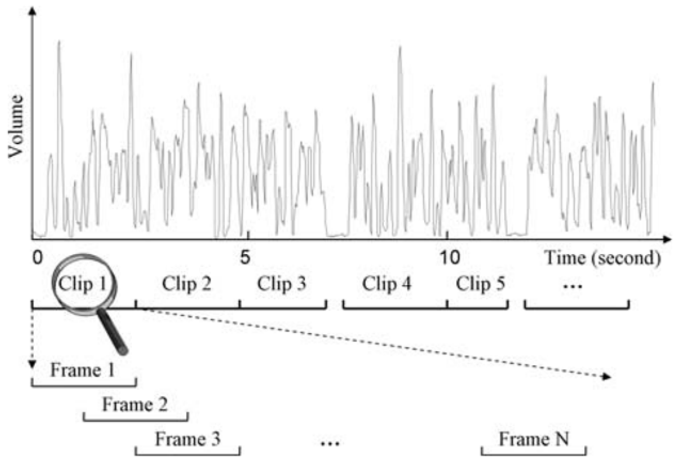


Figure 15: Clips and Frames

Frame level features are extracted by assigning numerical values to different properties in the signal at each frame. Whereas, the clip level features characterise how the frame-level features change over the clip. There are various frame level features that can be extracted from a digital signal, and generally these can be separated into two categories; time-domain features, which are computed directly from the audio signal, and frequency-domain features, which are derived from the Fourier transform of the audio signal. A Fourier transform converts the digital signal from the time domain to the frequency domain, which in more simple terms means that instead of showing how the amplitude of the waveform changes over time, it shows the amplitudes of the different frequencies that are present within the signal (See Figure 16 [40] for a visual breakdown).

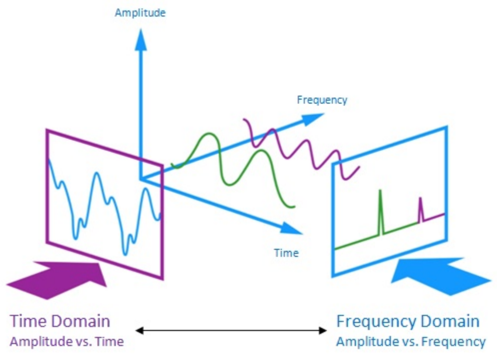


Figure 16: Time-domain vs. Frequency-domain

Some of the most commonly used frame-level features are described below:
<u>Time-Domain Features:</u>

- Energy - The energy of a frame corresponds to the total magnitude of the frame. This loosely equates to how loud the signal is in that frame.
- Zero-Crossing Rate - The zero-crossing rate of a frame is the rate of sign-changes of the signal within that frame. In a more simple sense, this feature is the number of times the signal changes value from positive to negative and vice versa.
- Entropy of Energy - The entropy of energy can be interpreted as a measure of abrupt changes in the energy level of an audio signal.

<u>Frequency-Domain Features:</u>

- Spectral Centroid - The spectral centroid of an audio signal is essentially the 'centre-of-gravity' of the spectrum produced by that signal.
- Spectral Entropy - Spectral Entropy is similar to the entropy of energy, but is done in the frequency domain.
- Spectral Flux - Spectral flux measures the spectral energy change between two successive frames.

These features provide low-level meta-data about the raw audio signal being processed. However, to extract more meaningful, higher-level information about the music being represented by the audio, these features have to be combined in a more complex way. Various techniques have been developed to infer musical features such as key, pitch, tempo, etc., which has its own uses and advantages. As part of this project, the main focus will be on the concept of beat and tempo detection in music. Beat is a fundamental element of all music that people can relate more easily with than other features such as key and pitch, which requires more musical experience to be able to accurately identify. Therefore, when trying to visualise music, the primary focus should be to effectively convey the tempo and the beats within the music.

3.3 Beat and Tempo Detection

Firstly, what exactly is meant by 'beat' and 'tempo'? Beat, in the theoretical sense, is the basic unit of measure in music. It can be described more simply as the pulse of a piece of music. Tempo, on the other hand, describes the speed of a piece of music, the tempo of music is usually measured as the number of beats per minute. As you can tell, these two terms are not mutually exclusive, meaning that the faster the tempo of a song, the faster the beat and vice versa.

Beat and tempo detection, like detection of any other feature, is challenging as it is not directly obvious from the audio waveform. Some forms of music emphasise beats using percussive instruments leading to peaks in the waveform, but this is not true for all forms of music, and sometimes can even be misleading as songs also tend to emphasise the off-beat times. There are various algorithms which try and find a general solution to this beat detection problem, but as mentioned earlier, the focus of this project will be on finding an algorithm that is light-weight but as accurate as possible in detecting beats and tempo in all genres of music.

BeatRoot: Simon Dixon

BeatRoot is the name given to an application created by Simon Dixon as an interactive beat tracking and metrical annotation system. BeatRoot allows users to display and manipulate musical beat data in a graphical interface and also provides audio feedback (music playback with percussive sounds to mark each beat in the music) [11]. However, the underlying beat-tracking algorithm that is being used within the application, which will be referred to as the BeatRoot algorithm from now on, will be the main focus of this project. BeatRoot uses a multi-agent architecture to simultaneously test multiple tempo hypotheses to determine the best fit. Figure 17 [11] shows the basic architecture of the BeatRoot algorithm. There are three main components to the BeatRoot algorithm; onset detection, tempo induction and beat tracking.

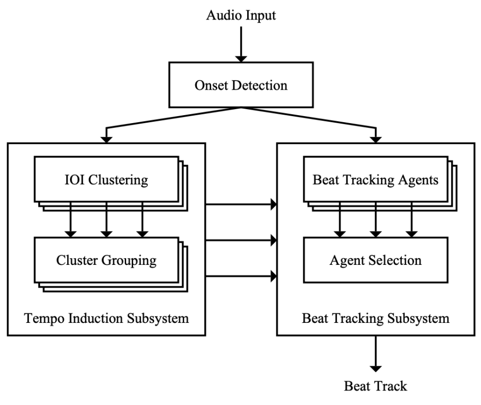


Figure 17: System Architecture of BeatRoot

Onset Detection

Onsets, in terms of audio, is defined as the beginning of a note or other sound. BeatRoot uses onset detection to find these musical notes to extrapolate the rhythmic times from an audio file. An onset detection function is a function whose peaks coincide with the times of music note onsets. Originally, in [10], the BeatRoot algorithm used a simple time-domain onset detection algorithm, which is outlined below:

1. The audio signal is passed through a first order high pass filter, which attenuates the low frequencies and passes only the higher frequencies.

2. The signal is then smoothed to produced an amplitude envelope. The amplitude envelope is calculated as the average absolute value of the signal within a window of that signal.

3. A 4-point linear regression model is then used to find the slope of the amplitude in the envelope.

4. Finally, a peak picking algorithm then finds the local maxima in the slope. Local peaks are rejected if there are peaks with a greater magnitude within a certain window of time or the peaks are below a specified threshold.

The algorithm above is loosely based on the method proposed by Schloss in [34], and as mentioned above, is done entirely in the time-domain. However, Dixon goes on to explain that this method is only well suited to music with drums, and that a frequency-domain onset detection method is required to work across various types of music. In an update of the algorithm, a spectral flux based onset detection function was used, due to its simplicity for programming, speed of execution and accuracy of correct onsets [11].

The spectral flux based onset detection function works by summing the change in magnitude along the frequency bins where the change is positive (energy is increasing). The first step for the spectral flux onset detection is to convert the audio signal from the time-domain to the frequency-domain using a Short-Time Fourier Transform (STFT). STFT calculates the Discrete Fourier Transform (DFT) value of a sliding window (short time segment) along the audio signal. Unlike a Fast Fourier Transform (FFT), which calculates the DFT the entire audio signal, STFT calculates how the DFT changes over time across the signal. Once this conversion is carried out, the spectral flux is calculated along the signal, which is then used by a peak picking algorithm to find the peaks. Dixon's paper [10] goes into further depth on how exactly the peak picking is carried out and the various thresholds and parameters used for the onset detection, but that level of complex understanding is not required for this project.

Tempo Induction

The next section of the BeatRoot algorithm is the tempo induction stage. The idea behind this stage is to use the onsets identified in the previous stage to determine a list of possible tempo hypotheses for the audio signal which can then be used by the next stage. The basic steps of the tempo induction algorithm are [10]:

1. Calculate the Inter-Onset Interval (IOI) between every onset against every other onset identified in the previous stage.

2. Use clustering to assign each IOI to a specific cluster. These clusters will represent our tempo hypotheses; using the average value of the cluster as the interval for beat times (which can be used to calculate the tempo in beats per minute):

$$tempo_{bpm} = (1/IOI) * 60$$

3. Once the cluster formation is complete, pairs of clusters whose intervals have drifted together are merged and the clusters are ranked. Clusters are ranked based on the number of elements (onsets) they contain and then adjusted for related clusters. The adjustment for related clusters is used to reflect the idea that for every tempo hypothesis that exists, there may also exist clusters which represent integer multiples and divisions of the beat. The adjustment is applied to the clusters by calculating the average of the related clusters' normalised intervals, weighted by their scores. The rankings are also further adjusted by adding the scores of the related clusters, weighted by the $f(d)$ (where d is the integer ratio of cluster intervals) given by:

$$f(d) = \begin{cases} 6 - d, & 1 \leq d \leq 4 \\ 1, & 5 \leq d \leq 8 \\ 0, & otherwise \end{cases}$$

Figure 18 shows the pseudo-code algorithm for tempo induction and Figure 19 shows a example case with calculated rankings.

Definitions

Events are denoted by $E_1, E_2, E_3, \ldots$
The inter-onset interval between events E_i and E_j is denoted by $IOI_{i,j}$
Clusters are sets of inter-onset intervals denoted by $C_1, C_2, C_3, \ldots$
$C_i.\text{interval} = \frac{\Sigma_{j,k}\{IOI_{j,k} \epsilon C_i\}}{|C_i|}$
$f(n)$ is the relationship factor
i, j, k, m, n are positive integer variables

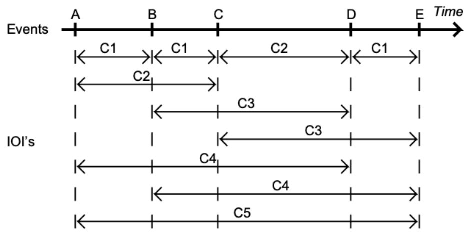
Algorithm

```
FOR each event $E_i$
    FOR each event $E_j$
        $IOI_{i,j} = |E_j.\text{onset} - E_i.\text{onset}|$
        Find $k$ such that $|C_k.\text{interval} - IOI_{i,j}| <$ ClusterWidth is minimum
        IF $k$ exists THEN
            $C_k := C_k \cup \{IOI_{i,j}\}$
        ELSE
            Create new cluster $C_m := \{IOI_{i,j}\}$
        END IF
    END FOR
END FOR

FOR each cluster $C_i$
    FOR each cluster $C_j (j \neq i)$
        IF $|C_i.\text{interval} - C_j.\text{interval}| <$ ClusterWidth THEN
            $C_i := C_i \cup C_j$
            Delete cluster $C_j$
        END IF
    END FOR
END FOR

FOR each cluster $C_i$
    FOR each cluster $C_j$
        IF $|C_i.\text{interval} - n * C_j.\text{interval}| <$ ClusterWidth THEN
            $C_i$.score:= $C_i$.score + $f(n) * C_j$.size
        END IF
    END FOR
END FOR
```

Figure 18: Tempo Induction Algorithm



```
C1.score = 2*3*f(1) + 2*f(2) + 2*f(3) + 2*f(4) + f(5) = 49
C2.score = 3*f(2) + 2*2*f(1) + 0 + 2*f(2) + 0         = 40
C3.score = 3*f(3) + 0 + 2*2*f(1) + 0 + 0              = 29
C4.score = 3*f(4) + 2*f(2) + 0 + 2*2*f(1) + 0         = 34
C5.score = 3*f(5) + 0 + 0 + 0 + 2*1*f(1)              = 13
```

Figure 19: Worked Example

Beat Tracking

Beat tracking is the third and final stage of the BeatRoot algorithm. The tempo induction phase, although finds a good estimate for the tempo, does not find the specific times of the beat occurrences in the audio file. This section is where the multi-agent architecture is used, where each agent in the system handles a specific tempo hypothesis and is able to assess themselves by predicting beat times and tracking how well they match with the events in the audio file. There are two main sections to the algorithm:

- Initialisation:
  For each tempo hypothesis, a group of agents are initialised to track the piece at that tempo for every event in the initial section of the audio file, with its first event coinciding with that beat time. Each agent is defined by a state, history and score. The state of an agent is its tempo hypothesis and its history is the sequence of beat times selected by the agent. The score of the agent is calculated using the salience of each event that is selected by the agent. Using the tempo hypothesis and the starting beat event, the agent is also able to predict a list of beat times that should occur within the audio file; this can be used by the agent to evaluate itself against every event it tracks.

- Main Loop:
  In the main loop, each event is processed by every agent. The predicted beat times mentioned in the initialisation stage is also given two windows of tolerance (inner and outer) which represents the extent to which each agent is willing to alter its prediction based on the events found. There are three possible scenarios that can occur when an agent processes an event:

  1. The most simplest case is when the event falls out of the tolerance windows of the predicted beat times; the event is simply ignored.
  2. When the event falls in the inner tolerance window the event is accepted by the agent. The agent's tempo is then updated by adjusting the value using the difference between the predicted beat time and the event time. The agent's score is also updated based on the salience of the event.
  3. In the case where the event falls in the outer tolerance window, the agent still accepts the event as a beat time. However, on the assumption it was incorrectly accepted, a duplicate agent is made where the event is not accepted. This allows both scenarios to be tracked. The scores of the agents at the end will determine the better choice.

The above description provides a brief overview of the algorithm, which is outlined in Figure 20. As mentioned earlier, a more in-depth explanation of the full BeatRoot algorithm can be found in Dixon's paper [10].

Initialisation

```
FOR each tempo hypothesis $T_i$
    FOR each event $E_j$ such that $E_j$.onset<StartupPeriod
        Create a new agent $A_k$
        $A_k$.beatInterval:= $T_i$
        $A_k$.prediction:= $E_j$.onset $+T_i$
        $A_k$.history:= $[E_j]$
        $A_k$.score:= $E_j$.salience
    END FOR
END FOR
```

Main Loop

```
FOR each event $E_i$
    FOR each agent $A_j$
        IF $E_i$.onset$-A_j$.history.last $>$ TimeOut THEN
            Delete agent $A_j$
        ELSE
            WHILE $A_j$.prediction$+Tol_{post} < E_i$.onset
                $A_j$.prediction := $A_j$.prediction$+A_j$.beatInterval
            END WHILE
            IF $A_j$.prediction$+Tol_{pre} \leq E_i$.onset$\leq A_j$.prediction$+Tol_{post}$ THEN
                IF $|A_j$.prediction $-E_i$.onset$| > Tol_{inner}$
                    Create new agent $A_k := A_j$
                END IF
                Error:= $E_i$.onset$-A_j$.prediction
                $A_j$.beatInterval:= $A_j$.beatInterval+Error$/$CorrectionFactor
                $A_j$.prediction := $E_i$.onset$+A_j$.beatInterval
                $A_j$.history:= $A_j$.history$+E_i$
                $A_j$.score := $A_j$.score$+(1-$relativeError$/2) * E_i$.salience
            END IF
        END IF
    END FOR
    Add newly created agents
    Remove duplicate agents
END FOR
Return the highest scoring agent
```

Figure 20: Beat Tracking Algorithm

4 SwarmPlayer: The Prototype

4.1 Introduction

SwarmPlayer is a music visualisation application, created as part of this project, which uses simulated swarm behaviour to visualise music. It was created with the intention of being a proof-of-concept prototype which encapsulates the ideas mentioned earlier in this project and creates a foundation for exploration into swarm based music visualisation. SwarmPlayer allows users to play audio files in the wav format and creates a visualisation for it using dots on the screen which emulate the swarm like behaviour found in animals and insects. The application focuses primarily on the concept of beat within music and implements the BeatRoot algorithm mentioned earlier.

4.2 Implementation

SwarmPlayer is implemented using Java; making use of the Swing toolkit for the GUI and the Clip interface to allow audio files to be read and processed. The BeatRoot algorithm was implemented in Java as part of the overall application by Dixon and has been adapted to work within SwarmPlayer. The basic structure of SwarmPlayer is outlined in Figure 21, which has 5 main components; Audio Processor, Audio Player, Event Timer, Model and Visualisation.

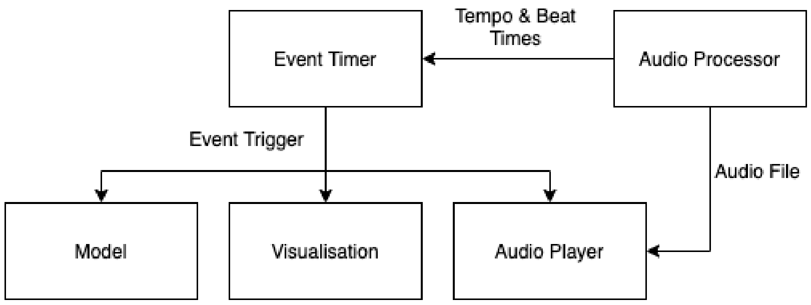


Figure 21: Architecture of SwarmPlayer

- The audio player sub-system is responsible for audio playback. It acts like a simple CD player; allowing a song to be loaded, played, paused/stopped and reset.
- When a file is selected, the audio processing sub-system pre-processes the file and carries out beat tracking (using BeatRoot) to find a tempo and the beat times for the audio file. This is then accessible by the event timer and the model sub-systems.
- The model sub-system is responsible for the modelling of the swarm. It maintains a list of boids[5] and carries out the swarm behaviour logic as well as applying visual mapping of beat times onto these boids.

[5]The term 'boid' is used in Reynolds' paper to refer to a single entity used in the simulation and will be used similarly in this report.

- The visualisation sub-system simply takes each boid from the model and draws them on the screen. It is also responsible for dealing with the button actions which allow the user to load, play, pause and reset the visualisation and the audio.

- The event timer sub-system acts as an over-arching control system which deals with the synchronisation between audio and visualisation. It triggers events which either update the world model or re-render the visualisation panel. The event timer is also responsible for handling button click events from the user to stop and start the audio.

4.2.1 Audio-Visual Mapping

An important factor that had to be considered during the implementation of the SwarmPlayer application was how the audio features were going to be mapped on to visual effects in the swarm simulation. As described previously, the BeatRoot algorithm provides two pieces of information about the audio file selected; the overall tempo of the piece and a list of beat times within the piece (also the salience of each beat, which will become useful later).

The tempo of the piece can be easily mapped to the maximum speed of each boid, as this would easily convey the speed of the piece by changing the speed of the boids in the simulation, so that the overall simulation would appear to move faster, the faster the tempo of the song is. In the implementation of SwarmPlayer, the function listed below is used to find a mapping between the allowed tempos for the BeatRoot algorithm to an integer value for the maximum speed of each boid:

$$maxSpeed = (\frac{tempo - 50}{150}) \times 20$$

This mapping function was determined empirically using music with varying tempos and sets the maximum boid speed once the file has been loaded in.

When coming up with a visualisation mapping for the beat times; a simple pulsating idea was used. This means that, for every beat time, the boids in the simulation would disperse away from each other for a small period of time before coming back together, creating a pulsing effect. This is shown in Figure 22 which are screenshots taken during a visualisation run in SwarmPlayer.

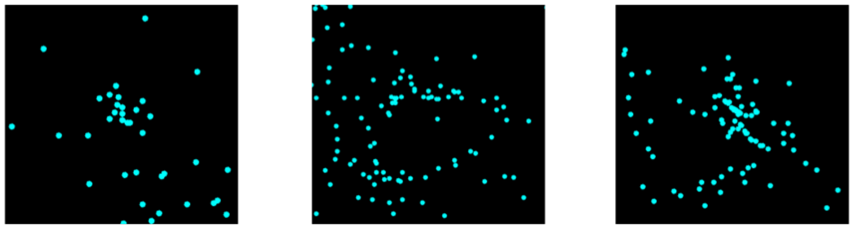


Figure 22: Pulse Effect

This pulsating effect can be obtained in the swarm simulation by manipulating the weighting coefficients used when combining the three behaviours required for swarming. By temporarily increasing the weighting of the separation behaviour and lowering the weighting of the cohesion behaviour, the boids in the simulation will be primarily heading in a direction away from each other, creating the firework-like explosion. Then, by switching the weightings of those two behaviours, an implosion effect can be achieved to finish off the overall pulse. However, once this was implemented, it was noticed that the time in between beats was too short for the full pulse effect to take place. Consequently, the boids were constantly separating away from each other, making the visualisation quite uninteresting. To remedy this, the list of beat times were filtered so that only the top $\sim 15\%$ of beat times were used; the ranking of the beat times were based on the salience value assigned by the BeatRoot algorithm.

4.2.2 The Toroidal World Model

The model sub-system as mentioned above is responsible for modelling the overall simulation. One important consideration to be made when modelling is the world model that the boids exist in. In a two dimensional world there are two main world ideas that can be used:

- Box World - The concept of a box world is quite simple; in the box world the edges of the 2D screen represent the edges of the world which trap the boids in a rectangular box. This type of world would require the boids to implement a wall avoidance type behaviour to stop them from going beyond the edges of the screen.

- Toroidal World - A toroidal world is sometimes also referred to as a 'wrap-around' world as the basic idea is that the edges of the screen wrap around to the opposing edge. So an object which travels towards and past the right edge of the screen would simply re-appear on the left side of the screen. Theoretically, this world model forms a torus shape as shown in Figure 23[15], which allows the boids to move around freely with no restrictions.

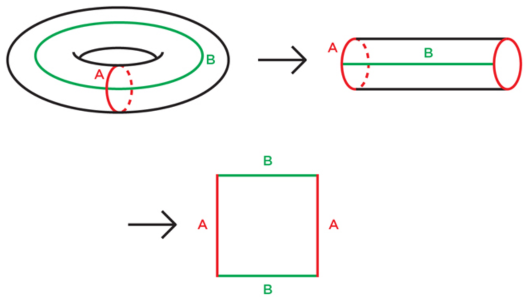


Figure 23: Torus to Rectangle Transformation

Although a box world creates a more contained visualisation world, this restriction affects the simulation of swarm behaviour as it bounds the world in which the boids can move and causes quite unnatural behaviour. Swarm behaviour in nature is usually found in quite vast open spaces, i.e. birds in an open sky and fish in a vast ocean, which means that having a more restrictive world does not naturally give rise to this swarm behaviour and forms quite erratic behaviour while trying to avoid the edges of the world. The implementation of a wall avoidance behaviour also raises the problem of balancing the swarming behaviour against the avoidance behaviour, as such a strict bounded world model would mean that the boids spend more time trying to avoid the edges rather than swarming with each other.

For this reason, the world model implemented as part of the SwarmPlayer simulation is a toroidal world. To ease the computational complexity, a simple 2D coordinate system is used for the implementation, where each boid is given an x and y value to represent its position on the screen. However, this means there are some further considerations that have to be made to the swarm behaviour equations described earlier, due to the edge cases. Each boid has to consider all of the neighbouring boids, within a certain perception radius, which becomes a problem when the boid reaches the edge of the screen as the neighbouring boids may be on the other side of the screen. Therefore, the traditional method for calculating distance between boids has to be altered to allow for this wrap-around world. Another crucial issue that arises due to the wrap-around world with 2D coordinates is when calculating the average position of the neighbouring boids in the cohesion behaviour, as the normal method of summing the positions would give an average position in the middle of the screen for edge cases. Therefore, this again has to be adjusted to factor in the wrap-around world.

Calculating Distance

Usually, when calculating the distance between two points: (x_1, y_1) and (x_2, y_2), you can use Pythagoras' equation:

$$distance^2 = (x_2 - x_1)^2 + (y_2 - y_1)^2$$

However, in a toroidal world, this equation does not always give the shortest possible distance as there are various possible directions which could be taken to measure distance in this wrap-around world. When considering the edge cases in 2D, nine possible cases for the distances have to be considered, shown in Figure 24. But by analysing the equation above, it is evident that by minimising the two components $(x_2 - x_1)$ and $(y_2 - y_1)$, the minimal distance between the points can be obtained. So, to simplify this problem, each dimension will be considered independently. Due to the torus shape of the wrap-around world, the furthest any two points can be along one dimension is half the maximum size of that dimension. So, considering the x dimension, in the case that $(x_2 - x_1)$ is less than half of the screen width (m), then the minimal distance has been found and no further calculation is required. However, in the case where this is not true, the side of the screen (left/right) which the principle boid (x_1) is on has to be determined. If it is on the right-hand side, the difference between x_1 and $x_2 + m$ needs to be calculated, so that the neighbouring boid is effectively 'moved' past the edge of the theoretical screen but closer to the principle boid. In the case that the principle boid is on the left-hand side of the screen, the neighbouring boid is simply 'moved' in the opposite direction by subtracting m from it. The overall algorithm in the x dimension therefore becomes:

```
1       diffx = x2 − x1
2       if(abs(x2 − x1) < m/2)
3       {
4           if(x1 < m/2)
5           {
6               diffx −= m
7           }
8           else
9           {
10              diffx += m
11          }
12      }
13      return diffx
```

A similar method can be used to calculate the minimal difference in the y dimension and then these differences can be substituted back into Pythagoras' equation mentioned above to find the minimal distance.

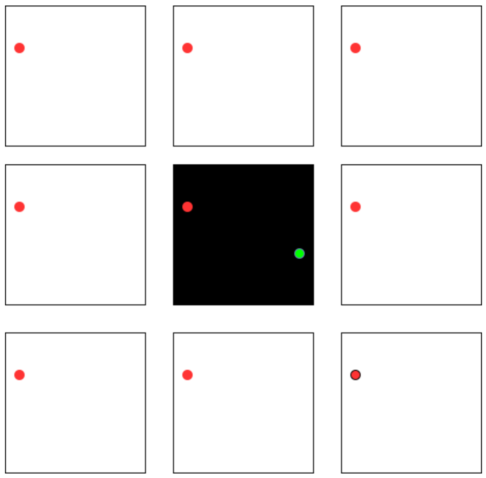


Figure 24: 2D Cases To Consider

Calculating Position

Calculating a wrap-around position poses a similar problem to that of calculating distance above. Therefore, a similar solution can be used to overcome this problem, but instead of focusing on the difference between the points, a new position needs to be created which puts the neighbouring boid closer to the principle boid. In the algorithm above, in lines 6 and 10, instead of updating the diffx value, a new $x3$ variable will be updated to hold the x value which produces the minimal distance. The algorithm now becomes:

```
diffx = x2 - x1
x3 = x2
if(abs(x2 - x1) < m/2)
{
    if(x1 < m/2)
    {
        x3 -= m
    }
    else
    {
        x3 += m
    }
}
return x3
```

Similar to calculating distance, this same method can be applied in the y dimension to find (x_3, y_3), which can then be used in the normal way to find the average position of a group of neighbouring boids.

4.3 GUI: Layout and Functionality

The SwarmPlayer application has a very simple GUI with only very limited functionality. Figure 25 shows a screenshot of SwarmPlayer whilst being run on a Macbook. There are two main components to the application; the toolbar and visualisation pane. The visualisation pane is where the swarm simulation is drawn and animated; there is no user interactivity allowed directly on the visualisation pane. The toolbar on the other hand has six sub-components:

- Swarm Mixer Button - Brings up a swarm mixer window that will be discussed later.
- Start/Pause Button - Allows the user to start or pause the visualisation and the audio. It is only enabled once an audio file is loaded in.
- Load/Re-Load Button - Allows the user to load an audio file in and re-load another if one is currently loaded. It will bring up a file selection window which will allow the user to traverse their file system.
- Reset Button - This button will reset the entire system so that any loaded audio file is cleared and the visualisation pane is also cleared.
- Progresss Bar - Shows the progress of the current track being played. It is not interactive and therefore does not allow users to adjust the position of the track.
- Playing Timer - Displays the progress of the audio as a numerical value. It also shows the overall time of the audio file that is loaded in.

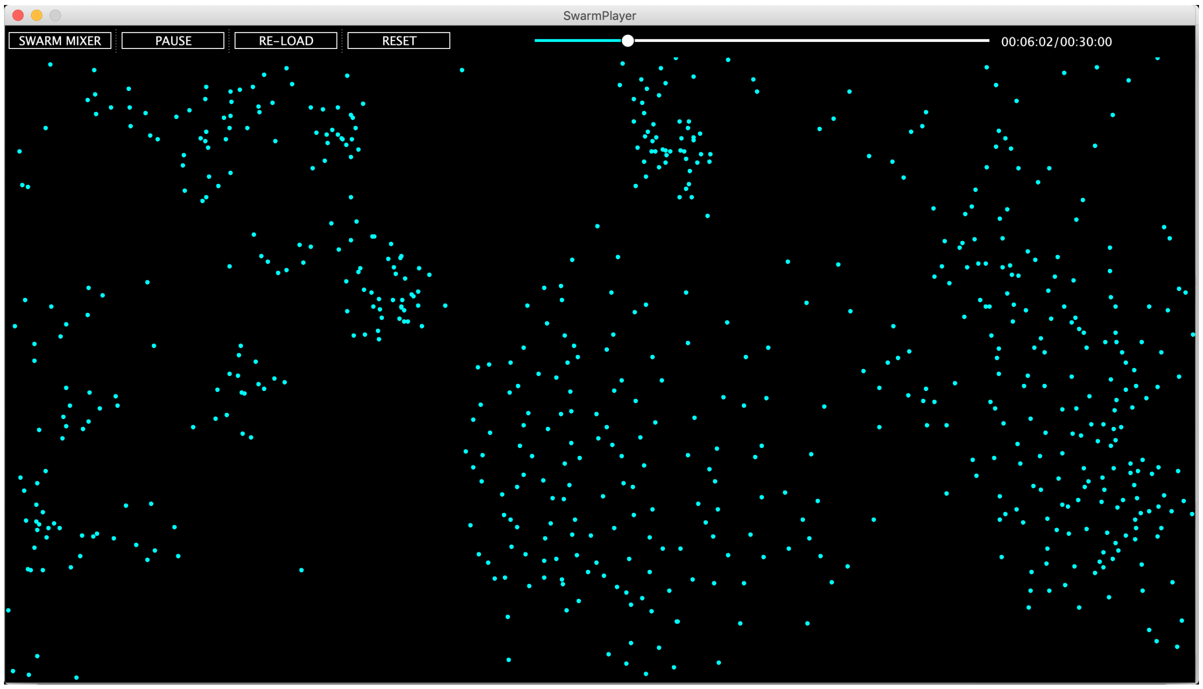


Figure 25: SwarmPlayer GUI

4.3.1 Swarm Mixer

The swarm mixer is an incredibly useful tool which allows the user more manual control over the visualisation being displayed. Figure 26 shows a screenshot of the window that appears when you click the swarm mixer button. This window mimics the usage of a traditional audio mixing board which has sliders that allow the users to control the levels of various audio sources. However, unlike audio mixers, the swarm mixer allows the user to control certain parameters in the swarm simulation:

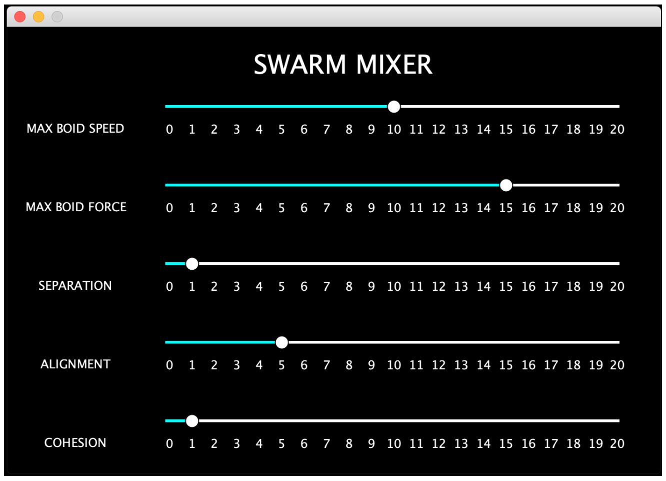


Figure 26: Swarm Mixer

- The max speed slider allows the user to control the maximum speed that each boid can travel at, and therefore control the overall speed of the simulation. As mentioned previously, this value is initially set once the audio file is loaded and is based on the tempo of the audio file.
- The max force slider allows the user to control how much of an effect one boid can have on another. A good balance between the speed and force is required for an effective visualisation, but the user is free to experiment using these sliders.
- The remaining three sliders allow the users to manually adjust the default values used for the weighting coefficients for the behaviours required for swarming; separation, alignment and cohesion. The slider values show a $\times 10$ scale of the coefficients used in the simulation (i.e. a 2 on the slider would output 0.2 as the coefficient value). These sliders unfortunately do not affect how the beat time visual mapping is carried out, which will still affect the values temporarily to create the desired pulsing effect but then return to the default values set by the user after.

4.4 Testing

As SwarmPlayer is intended as a prototype rather than a fully developed application, the testing done as part of the implementation was quite minimal. There were two main sections of testing which were carried out during and after the implementation of SwarmPlayer: unit testing and user testing. Testing was conducted with two main aims; ensuring that the underlying algorithms were working as expected and to obtain feedback on the visualisation approach proposed.

4.4.1 Unit Testing

Unit testing was carried out as part of the implementation to test the swarming algorithms in the simulation. These unit tests were created using JUnit, which is an open source unit testing framework for Java. Unit testing was specifically focused on five main points of functionality and the method for testing each of them is outlined below:

- Distance Measurement - As mentioned earlier, calculations in this wrap-around toroidal world are more complex and therefore needed to be tested. This was tested by checking that whether a boid is able to accurately filter a list of boids to find neighbouring boids within a certain perception distance. There were three test cases used, all in a 200x200 world with the principle boid in varying positions; centre of the screen, right edge of the screen and bottom-right corner of the screen. Then, a list of boids were given; five neighbouring the principle boid and five at various points not within the perception radius of the principle boid, and the test compared whether the list was accurately filtered down to the five neighbouring boids.
- Separation Behaviour - The separation behaviour was tested by providing a principle boid and a list of neighbours, and comparing the output direction against an expected output direction.

Three test cases were used for testing this behaviour. Each of the test cases used a 200x200 world with one principle boid in varying positions; centre of the screen, right edge of the screen and bottom-right corner of the screen.

- Alignment Behaviour - Alignment behaviour was tested using three test cases, with only a list of neighbours. A principle boid was not required for calculating the average velocity of the neighbouring boids. The tests compared the steering direction produced by the alignment behaviour function against an expected direction to determine its accuracy.

- Cohesion Behaviour - Cohesion behaviour was tested in a similar manner to separation and alignment behaviour. Three test cases were used, each in a 200x200 world with a principle boid in varying positions; centre of the screen, right edge of the screen and bottom-right corner of the screen, and five neighbouring boids. The tests compared the steering direction produced by the cohesion behaviour function against an expected direction to determine its accuracy. Testing the cohesion behaviour also ensured that the method for calculating the average position of a list of boids in the wrap-around world works accurately.

- Flocking/Swarming - Flocking/Swarming behaviour was also tested to ensure that the three behaviours listed above were combined together accurately. As only the combination of the behaviours was being tested, only one test case was used to determine whether the flocking function was working accurately. The test case used a 200x200 world with one principle boid and ten other boids; five of which was a neighbour to the principle boid.

All tests were confirmed as having passed, however, more extensive testing may be required if this application was to be scaled up.

4.4.2 User Testing

The user testing is perhaps the most important form of testing carried out on SwarmPlayer. User based testing gives an insight into the effectiveness of the proposed visualisation system. The proposed system for conducting this user testing was an online questionnaire with visualisation videos embedded as part of the questionnaire, which should allow a large number of users to participate. Unfortunately, due to limitations of not being able capture a video recording of the visualisation without causing lag, the user testing was done in a live situation where the user would watch the visualisation on a laptop and complete a questionnaire[6] on a mobile device.

Due to the restriction of having to conduct the questionnaire in-person with all the candidates, it proved difficult to obtain a large sample of users to evaluate the application. Overall, the evaluation was completed by five participants, which, although is not a large enough sample for statistical analysis to provide reliable answers, provided interesting qualitative feedback.

[6]The questionnaire used for the evaluation can be found at `https://forms.gle/RvjqKp1bw4x5yX6Q9`.

The first section of the questionnaire focuses on extracting background information about the participants, in particular, their experience with music in general and music visualisers. The participants' experience with music theory was varied, in that two out of five of them did have experience with music theory and the others did not. This supports the results of the evaluation as it provides good coverage of musicians and non-musicians when comparing and contrasting responses. The second question indicates that all participants were regular listeners of music, with all of them answering either "Often" or "Very Often" for the question. All of the participants stated that they had no previous experience with music visualisers. This is an interesting insight as many of us unknowingly deal with music visualisers on a daily basis in form of music videos. Although this wasn't mentioned as part of the examples of visualisers outlined earlier in the paper, music videos serve as manual form of music visualisation. This is something worth noting as exposure to this form of visualisation could unintentionally set unrealistic expectations of visualisers in the participants' mind, meaning the results of this evaluation may be skewed. The final question regarding the participants' opinion on whether music visualisers enhance the listening experience is redundant as all the participants outlined having no experience with music visualisers and therefore would be unable to provide any opinion for this question.

In the next few sections of the questionnaire, participants were presented an example visualisation with a short except of music and then asked to answer a few questions regarding what they watched. The participants were presented with four visualisations:

- Visualisation 1 - A short classical music excerpt with a tempo of approximately 120bpm.
- Visualisation 2 - Another classical music excerpt but with a faster tempo of approximately 150bpm.
- Visualisation 3 - A heavy rock song excerpt with a tempo of approximately 135bpm.
- Visualisation 4 - A short hip-hop excerpt with a tempo of approximately 100bpm.

The four short excerpts of music used for the test is included within the source code for SwarmPlayer, which is included as part of this project.

All of the participants agreed that the tempo was well conveyed in all of the visualisations and that the visualisations were compelling and immersive. However, the question regarding the conveyance of beat within the visualisation produced varying results. Majority of the participants agreed that visualisations 1 and 2 conveyed the beat of the music quite well but visualisations 3 and 4 were not as highly rated, with an average value of approximately 2 on the 5-point Likert scale. As stated earlier, all participants agreed that the visualisation was interesting and furthermore, four out of the five participants even agreed that the visualisation enhanced their listening experience.

The comments from the various parts of the questionnaire indicated one repeated point that were made by all participants; they all noticed that certain prominent beats from the audio were missed, especially in visualisation 3 where the visualisation doesn't seem to react at all until the louder segment kicks in. This comment doesn't necessarily reflect a failure of the SwarmPlayer system but rather a failure in the beat tracking algorithm. It goes to highlight one of the key difficulties within MIR and further emphasises the difficulty of what this project was trying to achieve. They also noticed that sometimes the swarms spread out so much that they reached an almost equilibrium state which meant the visualisations became less interesting. This would be something that can be remedied by including further manipulations of the swarms in a future update of SwarmPlayer so that the swarms are constantly in flux and never reaches an equilibrium state. An intriguing comment, made as part of the evaluation, was that in visualisation 4 the swarms seemed to 'bounce' to the beat just like dancers would when dancing to that type of music. This emphasises the success of this project and supports the use of swarms as a visualisation technique.

5 Conclusions and Future Work

5.1 Time Complexities and Optimizations

One thing that has not been considered much in this project is the notion of computational complexity. As the SwarmPlayer application was intended as a prototype, considerations for optimizing performance were not prioritised. However, since a simple model where all boids are stored in a globally accessible list is used, the complexity of the swarm simulation is in the order of the square of the swarm's population ($O(n^2)$ where n is the number of boids in the simulation). This complexity arises due to each boid having to compare itself against every other boid in the world to determine its neighbours. This is a crucial concern as to achieve an effective swarm visualisation, upwards of 400 boids are required; otherwise, the boids would be too sparsely distributed to form interesting patterns. The issue that arises from this complexity can be seen within the SwarmPlayer implementation, which currently uses 700 boids. The simulation within SwarmPlayer starts to lag when the number of boids is too high.

A simple solution to this problem would be to separate each boid so that they work in their own processor/processing thread. This would make determining its neighbours more of a complex problem but would make complexity linear with respect to the size of the swarm ($O(n)$). However, another major issue with this would be the high processor usage, each processor/processing thread would be locked off for each boid and would be performing very simple calculations, making inefficient use of the computer's resources.

Another solution is to try and keep n very small without reducing the swarm size. This basically means reducing the number of boids that each boid has to assess to determine its neighbours. This can be achieved by using a technique known as bin-lattice spatial subdivision [33]. In bin-lattice spatial division, the screen is divided into equal box shaped segments called 'bins'. At the start of the simulation, the boids are distributed into the various bins based on their initial positions, and after each movement, their bin membership is updated if they move out of their bin. A quad-tree structure can also be used to achieve similar results, where the quad-tree would be re-created at each simulation step to track the movements of the boids. Although further calculations need to be done for updating bin memberships or reforming quad-trees, this far outweighs the computational complexity reduced by using these data structures when simulating large swarm populations.

5.2 Further Ideas

For this project, the simulation of swarm behaviour was restricted to the three group behaviours outlined by Reynolds to achieve flocking in [31]. However, in a later paper from Reynolds, there were several other behaviours such as seeking, fleeing and flow-field following included which create a more realistic simulation. Although these behaviours are not required in a basic swarm simulation, the implementation of these behaviours open up more possibilities for audio-visual mappings.

SwarmPlayer also only currently detects beat and tempo within audio files, but research in MIR has opened up possibilities of extracting various other features from audio such as pitch, key, mood, etc. These can map onto various aspects in the simulation as it currently is, such as boid size, boid colour, or even by manipulating the weighting coefficients in some other way to affect the swarming behaviour.

5.3 Conclusion

The primary aim of this project was to unify the research conducted within the field of music visualisation, MIR and swarm behaviour simulation to create a unique and appealing music visualisation approach. SwarmPlayer was developed as part of this project to provide a concrete proof-of-concept prototype that brings to life the visualisation approach proposed. Although the evaluation and testing of this approach is quite minimal, it further supports the idea of using swarm based simulation for music visualisation. From the evaluation it is evident that the visualisation technique is engaging and also conveys music features quite effectively. At the start of this project we briefly explored the profound effects that music visualisers can have on the listening experience and as part of the evaluation, majority of the participants agreed that this visualisation approach did enhance their listening experience. However, the extent to which and the exact effects of the visualisation on the listening experience will require further investigation to understand.

Another aspect that can be further studied is the mapping between the beat and the visual representation of it in the swarm. SwarmPlayer uses a simple pulsing effect to convey the beats within the audio, which was found to be problematic when reacting to all beats in the music as the time in-between beats does not allow for natural swarming to occur and therefore produces unappealing visualisations. Further research into alternate techniques for conveying beat or effectively filtering out less prominent beats in the music could further enhance the visualisation.

The BeatRoot algorithm used as part of SwarmPlayer provided an efficient and lightweight algorithm. However, the accuracy of the algorithm is questionable. Therefore, a more in-depth test of the algorithm and comparison against possible alternatives would be required to choose the best fitting algorithm. An interesting concept that has not been explored by this project would be to carry out real-time processing of the audio, so that the audio is processed while being read in and the visualisation is produced at the same time.

Overall, this project was successful in providing a foundation in using swarm based simulations for their visual appeal rather than for optimisations. This project also re-visits the use of music visualisers and brings a fresh perspective which could make music visualisers more popular. The implementation of SwarmPlayer serves as a base for further exploration into swarm based music visualisations, with the possibility of implementing more behaviours or being able to convey more audio features within the simulation to make it more captivating.

6 References

[1] *Audio Features (Audio Processing) (Video Search Engines)*. URL: `http://what-when-how.com/video-search-engines/audio-features-audio-processing-video-search-engines/` (visited on 03/27/2020).

[2] Bruce M. Blumberg and Tinsley A. Galyean. "Multi-level direction of autonomous creatures for real-time virtual environments". en. In: *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques - SIGGRAPH '95*. Not Known: ACM Press, 1995, pp. 47–54. ISBN: 978-0-89791-701-8. DOI: `10.1145/218380.218405`. URL: `http://portal.acm.org/citation.cfm?doid=218380.218405` (visited on 03/16/2020).

[3] Fran Board. *Audio-visual interactions: How images affect our perception of music — sound.surf*. 2019. URL: `https://sound.surf/news/audio-visual-interactions` (visited on 03/05/2020).

[4] Marilyn G. Boltz. "Musical Soundtracks as a Schematic Influence on the Cognitive Processing of Filmed Events". In: *Music Perception: An Interdisciplinary Journal* 18.4 (2001). Publisher: University of California Press, pp. 427–454. ISSN: 07307829, 15338312. DOI: `10.1525/mp.2001.18.4.427`. URL: `www.jstor.org/stable/10.1525/mp.2001.18.4.427`.

[5] Marilyn G. Boltz, Brittany Ebendorf, and Benjamin Field. "Audiovisual interactions: The impact of visual information on music perception and memory". In: *University of California Press Journals* 27.1 (2009), pp. 43–59.

[6] Manuele Brambilla et al. "Swarm robotics: a review from the swarm engineering perspective". en. In: *Swarm Intelligence* 7.1 (Mar. 2013), pp. 1–41. ISSN: 1935-3820. DOI: `10.1007/s11721-012-0075-2`. URL: `https://doi.org/10.1007/s11721-012-0075-2` (visited on 03/14/2020).

[7] Chris Cannam, Christian Landone, and Mark Sandler. "Sonic visualiser: an open source application for viewing, analysing, and annotating music audio files". en. In: *Proceedings of the international conference on Multimedia - MM '10*. Firenze, Italy: ACM Press, 2010, p. 1467. ISBN: 978-1-60558-933-6. DOI: `10.1145/1873951.1874248`. URL: `http://dl.acm.org/citation.cfm?doid=1873951.1874248` (visited on 03/05/2020).

[8] Annabel J. Cohen. "Music as a Source of Emotion in Film". In: *Handbook of Music and Emotion: Theory, Research, Applications*. Ed. by Patrik N. Juslin and John Sloboda. Oxford University Press, 2011.

[9] *Cycling '74 Max*. URL: `https://cycling74.com/products/max/` (visited on 03/11/2020).

[10] Simon Dixon. "Automatic Extraction of Tempo and Beat From Expressive Performances". In: *Journal of New Music Research* 30 (Aug. 2001). DOI: `10.1076/jnmr.30.1.39.7119`.

[11] Simon Dixon. "Evaluation of the Audio Beat Tracking System BeatRoot". en. In: *Journal of New Music Research* 36.1 (Mar. 2007), pp. 39–50. ISSN: 0929-8215, 1744-5027. DOI: `10.1080/09298210701653310`. URL: `http://www.tandfonline.com/doi/abs/10.1080/09298210701653310` (visited on 03/30/2020).

[12] *Flash Mob Definition*. en. Library Catalog: www.lexico.com. URL: `https://www.lexico.com/definition/flash_mob` (visited on 03/19/2020).

[13] *Flash Mob Image*. en-US. Library Catalog: artofthemooc.org. URL: `https://artofthemooc.org/wiki/history-and-origins-of-the-flashmob/` (visited on 03/19/2020).

[14] Simon Frith. "Music and everyday life". en. In: *Critical Quarterly* 44.1 (2002), pp. 35–48. ISSN: 1467-8705. DOI: `10.1111/1467-8705.00399`. URL: `https://onlinelibrary.wiley.com/doi/abs/10.1111/1467-8705.00399` (visited on 03/05/2020).

[15] *From Hyperbolic Geometry to Cube Complexes and Back.* en. URL: https://www.quantamagazine.org/from-hyperbolic-geometry-to-cube-complexes-and-back-20121002/ (visited on 04/03/2020).

[16] Joe Futrelle and J. Stephen Downie. "Interdisciplinary Communities and Research Issues in Music Information Retrieval". In: *ISMIR.* 2002.

[17] Alan Graves, Chris Hand, and Andrew Hugill. "MidiVisualiser: interactive music visualisation using VRML". en. In: *Organised Sound* 4.1 (Jan. 1999), pp. 15–23. ISSN: 13557718. DOI: 10.1017/S135577189900103X. URL: http://www.journals.cambridge.org/abstract_S135577189900103X (visited on 03/04/2020).

[18] Michael Hahn. *Sound and Vision: Music Visualization Lets You See Your Sounds | LANDR Blog.* 2019. URL: https://blog.landr.com/music-visualization/ (visited on 03/05/2020).

[19] Adam Hayes. *Sine Wave Definition.* en. Library Catalog: www.investopedia.com. URL: https://www.investopedia.com/terms/s/sinewave.asp (visited on 03/28/2020).

[20] Dirk Helbing, Joachim Keltsch, and Péter Molnár. "Modelling the evolution of human trail systems". en. In: *Nature* 388.6637 (July 1997). Number: 6637 Publisher: Nature Publishing Group, pp. 47–50. ISSN: 1476-4687. DOI: 10.1038/40353. URL: https://www.nature.com/articles/40353 (visited on 03/11/2020).

[21] R. Hiraga, F. Watanabe, and I. Fujishiro. "Music learning through visualization". In: *Second International Conference on Web Delivering of Music, 2002. WEDELMUSIC 2002. Proceedings.* ISSN: null. Dec. 2002, pp. 101–108. DOI: 10.1109/WDM.2002.1176199.

[22] *How to trip out with the iTunes visualizer.* en. Library Catalog: www.macworld.com. May 2016. URL: https://www.macworld.com/article/3074173/how-to-trip-out-with-the-itunes-visualizer.html (visited on 03/05/2020).

[23] Hans Jenny. "Cymatics: A Study of Wave Phenomena and Vibration". en. In: (1968), p. 135.

[24] Michael Kassler. "Toward Musical Information Retrieval". In: *Perspectives of New Music* 4.2 (1966). Publisher: Perspectives of New Music, pp. 59–67. ISSN: 0031-6016. DOI: 10.2307/832213. URL: https://www.jstor.org/stable/832213 (visited on 03/21/2020).

[25] John McGowan, Grégory Leplâtre, and Iain McGregor. "CymaSense: A Real-Time 3D Cymatics-Based Sound Visualisation Tool". en. In: *Proceedings of the 2016 ACM Conference Companion Publication on Designing Interactive Systems - DIS '17 Companion.* Edinburgh, United Kingdom: ACM Press, 2017, pp. 270–274. ISBN: 978-1-4503-4991-8. DOI: 10.1145/3064857.3079159. URL: http://dl.acm.org/citation.cfm?doid=3064857.3079159 (visited on 03/07/2020).

[26] Windows Media Player. *Winter Visualization, Snow Flakes.* 2007. URL: https://sec.ch9.ms/ecn/c4fcontent/migration/6821379/capture1.png (visited on 03/05/2020).

[27] David Moffat, David Ronan, and Joshua D. Reiss. "An Evaluation of Audio Feature Extraction Toolboxes". en. In: (2015), p. 7.

[28] Meinard Müller. *Fundamentals of Music Processing: Audio, Analysis, Algorithms, Applications.* en. Google-Books-ID: HCI_CgAAQBAJ. Springer, July 2015. ISBN: 978-3-319-21945-5.

[29] *Music Visualisation Definition.* en. Library Catalog: www.merriam-webster.com. URL: https://www.merriam-webster.com/dictionary/music+visualization (visited on 03/11/2020).

[30] Suranga Chandima Nanayakkara et al. "Towards building an experiential music visualizer". In: *2007 6th International Conference on Information, Communications Signal Processing.* ISSN: null. Dec. 2007, pp. 1–5. DOI: 10.1109/ICICS.2007.4449609.

[31] Craig W. Reynolds. “Flocks, Herds, and Schools: A Distributed Behavioral Model”. en. In: (1987), p. 13.

[32] Craig W. Reynolds. *Abstract Steering Behaviors For Autonomous Characters*. 1999.

[33] Craig W. Reynolds. “Interaction with Groups of Autonomous Characters”. In: *Game Developers Conference* 21 (July 2000).

[34] W. Schloss. “On the automatic transcription of percussive music [microform] : from acoustic signal to high-level analysis /”. In: (Jan. 1985).

[35] Evelyn Shaw. “THE SCHOOLING OF FISHES”. In: *Scientific American* 206.6 (1962). Publisher: Scientific American, a division of Nature America, Inc., pp. 128–141. ISSN: 0036-8733. URL: `https://www.jstor.org/stable/24936575` (visited on 03/11/2020).

[36] Richard P. Smiraglia. “Musical Works as Information Retrieval Entities: Epistemological Perspectives”. In: *ISMIR*. 2001.

[37] *Swarm Definition*. en. Library Catalog: dictionary.cambridge.org. URL: `https://dictionary.cambridge.org/dictionary/english/swarm` (visited on 03/11/2020).

[38] Michael Taenzer, Burkhard C. Wünsche, and Stefan Müller. “Analysis and Visualisation of Music”. In: *2019 International Conference on Electronics, Information, and Communication (ICEIC)*. ISSN: null. Jan. 2019, pp. 1–6. DOI: `10.23919/ELINFOCOM.2019.8706365`.

[39] Unity Technologies. *Unity Real-Time Development Platform | 3D, 2D VR & AR Visualizations*. en. Library Catalog: unity.com. URL: `https://unity.com/` (visited on 03/11/2020).

[40] John VE6EY. *Signal Analysis for a Morse Decoder*. en-US. Library Catalog: play.fallows.ca Section: Ham Radio. Jan. 2017. URL: `http://play.fallows.ca/wp/radio/ham-radio/signal-analysis-morse-decoder/` (visited on 03/28/2020).