A.C.R.O.N.Y.M. - A Computational Re-Orchestration to Nuance Your Music

Hussein Elgridly
Supervisor: Iain Phillips
Individual Project Report for MEng Computing, Imperial College London

he04@doc.ic.ac.uk

27th June 2008

Abstract

Sound design in computer games is often an afterthought, with more effort being put into the more obvious areas such as graphics and AI. However, good sound and music design can add atmosphere to a game world, allowing the player to make an emotional investment in the game and thereby enriching the experience. The current mechanism for changing the mood of background music in games is to use a crossfade between two disparate tracks, which often gives a very jarring musical transition. A system which could dynamically modify the properties of a *single* music track would therefore be able to create smoother musical transitions, if it were possible to quantify the emotional effects of various musical modifications. This paper researches the possibility of creating such a system and presents ACRONYM, a music engine capable of dynamically modifying a MIDI file to create emotion, as well as the short demo game, *Gun Man: Struggle Eternal*, showcasing the possible use of ACRONYM in computer games.

Acknowledgements

I would like to thank my supervisor Iain Phillips, for guidance and feedback throughout the course of the project, as well as for sharing musical insights which had long since departed from my brain. Special thanks go to Steven Livingstone for discussing his music psychology research with me, and allowing me to reproduce parts of his work; and to Andrew Brown, maintainer of jMusic, without which my project would have been much more difficult. Credit goes to Tim Goodwyn for providing some test files and a couple of code snippets, to Andrew Sayers for help debugging some problematic code, and to Nathaniel Johnston for helping a complete stranger get his head around matrix maths, even if it never ended up being used. Finally I would like to apologise to my housemates Steve, Aleks, and Ed, for putting up with my incessant cries of "it's not working!", and to thank my girlfriend Heidi for politely pretending to care about a project that doesn't interest her in the slightest.

Artists

Chloe Bacon - the *Gun Man* animated sprite, boxes and barrels
Sam Stancill - backgrounds and other scenery objects

Testers

Chloe Bacon	Adrian Bennett	Steve Bilton
Oisin Challoner	Richard Drury	Heidi Fetula
Tim Goodwyn	Gordon Mackenzie	Martin-Patrick Molloy
Aleks Ponjavic	Andrew Sayers	Denis Sheeran
Maxim Zverev		

Contents

Chapter 1

Introduction

1.1 Motivation

From the very early days of *Space Invaders* [52] and *Pac-Man* [35] with their whizzing, beeping soundtracks to games of the modern age with fully orchestrated soundtracks such as *Super Mario Galaxy* [41], music has been a core part of computer games. Indeed, game music is often remembered as fondly as film soundtracks – the theme from *Tetris* [42] being one of the most recognised pieces of game music in the world. However, the games industry is still in its infancy and developers have only recently begun exploring the subtle and interesting effects that good sound design can have on the player. On the whole, there is still plenty of room for improvement; take, for instance, a relatively common sequence of events in a computer game:

1. The game is playing an arbitrary piece of background music, appropriate to the situation: for instance, the player is sneaking around an abandoned building, and there is 'tension' music playing.
2. An 'interesting' event happens in the game world: in our scenario, the player is spotted by an enemy.
3. This event sets off an in-game trigger to switch from the current music track to the 'action' music track. This is done with a simple crossfade.
4. Seconds later, the player kills the enemy; another trigger fires, and the music crossfades back to the 'tension' track, returning to the beginning of the piece.

The end result of this is a sudden, jarring transition to a second track which is musically very different – and then back again – over the course of just a few seconds. Repeated many times over the course of a game (or even a level), this rapidly becomes incredibly irritating to the player and can ruin the game experience. Surprisingly, poor musical transitions such as this are common in modern computer games, despite their negative effect on the player.

In recent years a trend for music-based games has appeared. These typically handle music in a more sophisticated manner, relative to the object of the game:

REZ In REZ, the player's avatar moves on a predetermined path through the game world, aiming and firing to shoot down enemies as they are approached. In each level, a basic trance music track plays and is embellished by trills and extra drum beats, which are generated by the player's actions. It is important to note though that the player's actions are locked to the beat of the music, so a shot may be delayed slightly for the sake of keeping to the rhythm.

Dance Dance Revolution *DDR* and its many spin-offs are dancing games. As the music plays, arrows move down the screen; when an arrow reaches the bottom of the screen, the player must step on the corresponding arrow on the pressure-sensitive pad at his feet. Points are scored for accurate timing and 'combos' – multiple steps performed correctly and within the allotted timeframe.

Guitar Hero / Rock Band These games put the players in the role of musician: each is equipped with an instrument-shaped controller, and on-screen graphics indicate when the notes on each instrument are to be played. Failure to 'play' an instrument skilfully enough results in the instrument track cutting out and the on-screen audience booing.

While these games immerse the player in the music, it is the music which dictates the player's actions – the gameplay is merely glorified button-pushing. *REZ* is the closest of these to link the background music to the player's actions, although it does fall short in limiting the timing of the player's actions. In short, as will become clear by the end of Section 2.2, games are failing to harness the power of music effectively: at best, this results in musically uninteresting experiences for the player; at worst, in intense irritation.

1.2 Contributions

This project aims to enhance the experience of computer games by providing a system for dynamically adapting the game's music to the in-game situation the player finds himself in. A better matching of the background music to the gameplay will provide a more atmospheric and memorable experience, increasing the player's enjoyment by providing the same musical 'cues' with seamless, instead of sudden, transitions. This report presents the following:

- A history of music in computer games (Section 2.1) and an analysis of the successes and failures in using music in games (Section 2.2);
- An overview of music psychology (Section 2.3) and the correlations between various musical properties and perceived moods (Chapter 4);

- A music engine, ACRONYM (Chapter 5). ACRONYM is capable of:
  - Taking as input a set of musical 'transformations' and a MIDI file, and outputting a variation of the MIDI file with the transformations applied;
  - Performing such an adaptation in real-time, while the MIDI file is playing;
  - Most importantly, taking a 'target emotion' instead of a set of transformations as input, and calculating the required transformations to achieve the target emotion before applying them in real-time as described above.
- ACRONYM also provides a standardised API for game developers wishing to use its features (Section 5.2.1).
- A graphical tool to control ACRONYM, the *Standalone Music Adaptation Tool* (SMAT, Section 6.2). SMAT allows the user to select the target emotion using a graph known as the 2DES, as well as allowing the user to specify the required transformations directly;
- A short 2D platformer game, *Gun Man: Struggle Eternal* (Section 6.3), to demonstrate a potential use of ACRONYM in a video game. The game features:
  - Parallax scrolling backgrounds to simulate character movement;
  - Collision detection between the player avatar, obstacles, enemies, and weapon fire;
  - Dynamic music using ACRONYM, reacting to both the player's location in the game world and the number of enemies currently on screen.
- A set of experiments designed to test ACRONYM's ability to alter the mood of a piece (Chapter 7).

Appendices A and B provide introductions to music theory and the MIDI specification respectively; those unfamiliar with either are invited to refer to them as necessary while reading this report in order to gain a clearer understanding.

We begin by presenting a background to game music, and an analysis of past lessons and related research.

Chapter 2

Background

This chapter will provide some background into the history of computer games and how their use of sound and music progressed as the industry gained experience. It will also examine the current state of music in computer games, highlighting a few games which use music to their advantage and analysing the deficiencies in the use of music in the majority of modern video games. Finally, we look to the future and the benefits better music in games can potentially bring.

2.1 A Brief History of Music in Games

1952 – 1982: The Early Years

In 1952, arguably the first computer game (a noughts and crosses simulator) was written for the EDSAC computer by A.S. Douglas as an illustration for his PhD in human-computer interaction at the University of Cambridge. *OXO* [10] played a perfect game against a human player, but despite this promising early start it was another two decades before *Pong* [2] achieved the widespread popularity necessary to launch computer games into the public eye. *Pong* made a 'beep' sound through the arcade machine's amplified mono speaker whenever the ball hit a paddle – one of the first recorded sounds in a computer game. It was a huge success – the initial prototype was placed in a bar and stopped working because the machine was overstuffed with coins. As a result of this success, Atari was founded – a driving force in the games industry that still exists to this day.

1975 saw Midway Games importing *Gun Fight* [34] from Japan, complete with gunshot sounds. They continued this tactic with considerably more success in 1978, this time importing *Space Invaders*. As the aliens move closer to the bottom of the screen, the audio's tempo increases, providing extra tension for the player. In the same year, the Magnavox Odyssey2 games system was released, featuring programmable cartridges that allowed games to have unique sound and music.[1] *Major League Baseball* [33] appeared the following year, with the first digitised voice in a game announcing 'strike', 'ball' etc.

In 1980 *Pac-Man* was released. To this day the opening theme is instantly recognisable and has become a pop-culture phenomenon. Even then, attention was so heavily focused on the game that when musicians Jerry Buckner and Garry Garcia spoofed Ted Nugent's song *Cat Scratch Fever* with *Pac-Man Fever*, the single reached #9 in the US charts.

1982 – 1993: Background Music Begins

Thirty years after the creation of *OXO*, songs by the rock 'n' roll band Journey were digitised into the game *Journey Escape* [9] for the Atari 2600. While games like *Space Invaders* and *Asteroids* [3] featured thumping backbeats as an accompaniment to gameplay, *Journey Escape* heralded the era of true background music in games. It was swiftly followed in 1983 by *Spy Hunter* [4] and *Dragon's Lair* [7], two of the first games to incorporate stereo sound.

However, the console that revolutionised background music in games was the NES (Nintendo Entertainment System), released in 1985. It boasted five sound channels, one of which was capable of playing PCM sound, and was the system on which both *Tetris* and *Super Mario Bros.* [36] were released. Both of these games were revolutionary: *Tetris* featured a repetitive yet mesmerising Russian-inspired piece of background music which is recognisable to this day; *Super Mario Bros.* had a soundtrack which varied with each level of the game. Nintendo continued with its success, releasing *The Legend of Zelda* [38] and *Final Fantasy* [37] for the NES in 1987. *Final Fantasy* in particular is widely known for its sweeping cinematic musical scores, and is currently on its 12th sequel, not including multiple spin-offs.

In 1989 Sega released *Michael Jackson's Moonwalker* [47] for the Genesis[2], offering digitised versions of the artist's songs, in much higher fidelity than *Journey Escape* featured. 1991 saw the release of the SNES (Super NES) and the pioneering game *ActRaiser* [11] for the same system. The Japanese role-playing game is credited for having an orchestrated, symphonic score, a level of musical detail not seen before. *Streets of Rage* [48] in the same year took advantage of the Genesis hardware, including some innovative uses of stereo effects. 1991 also saw the Tokyo Philharmonic orchestra playing arrangements of popular computer game soundtracks in the first of five Orchestral Game Concerts. The concept would not leave Japan for over a decade.

1993 – 2003: CD Quality Music

Sonic CD [49], released in 1993 for the Sega CD system, was the first game to make the jump to a CD-quality soundtrack. This was taken very seriously by Sega: the credits listed multiple composers, arrangers, and mixers, and even individual musicians. The release of the Sony PlayStation

[1]Up until the release of the Odyssey2, games were limited to the sound palette on the hardware they were running on.

[2]Called the MegaDrive in Europe.

in 1995 gave game developers the opportunity to work with 24 sound channels, which provided CD-quality audio as well as inbuilt support for digital effects such as reverb and looping.

In 1996 it started becoming the norm for famous musicians to provide background music for games: *Wipeout XL* [44] used songs from techno music artists The Chemical Brothers (amongst others) as its soundtrack, and the soundtrack for *Quake* [20] was specially composed by Trent Reznor of Nine Inch Nails fame.

Music-making in games first took off in 1998 with *The Legend of Zelda: Ocarina of Time* [39], released on the Nintendo 64: playing an ocarina transported the player you to new worlds. The game also featured a musical puzzle which required the player to follow the bass line of a piece of music to make it through uncharted territory. *Dance Dance Revolution* [25] (DDR) was also released in 1998, requiring players to 'dance' on pressure-sensitive floor pads in time to the music.

In 2000 Io Interactive took the step of commissioning the Budapest Symphony Orchestra to score the soundtrack for *Hitman* [21]. By 2001 video game music had the mainstream, with Internet radio stations dedicated to playing video game music all day, every day. *Super Smash Bros. Melee* [40] was released, including cameo appearances from many famous video game characters, complete with their musical themes rewritten and mixed together. In *REZ* [50], also released in 2001, sound effects were perfectly timed to coincide with beats in the background music – the first of many games to make music an integral part of the gaming experience.

2003 – Present: Musical Games

In recent years there have been a series of computer games where music has been the 'point' of the game. The first of these was *Karaoke Revolution* [26], released in 2003. It included a USB microphone and rewarded gamers with points if they sung within prescribed thresholds of timing and pitch. In 2005 the first *Guitar Hero* [17] game was released, bundling a guitar-shaped controller with the game. In the game, players simulate playing the guitar in various rock songs from the 1960s to present. It has sold 1.5m copies to date; its sequel, released the following year, has sold 3.1m copies. At the time of writing, *Rock Band* [18] has extended the concept behind *Guitar Hero* to a full band, using guitar, drum, and microphone peripherals. It costs $170 (including all controllers) and sold well over 380,000 units in its first month, despite being a US-only release at that point.

Concerts dedicated entirely to video game music are also becoming more common: 2003 saw the tradition established of a Symphonic Game Music Concert at the Games Convention fair in Leipzig, and in 2006 the first tour of *PLAY! A Video Game Symphony* was a massive success, touring the world to sellout shows.

While not the first game to have a song specially written for it, 2007's *Portal* [53] has rave reviews, in part due to the humorous song sung by one of the game's characters during the credit sequence. *Super Mario Galaxy* features an orchestrated soundtrack played at precisely the right tempo to fit in with the protagonist's movements, and sound effects are also timed to fit in with the music appropriately. The game has won *Edge* magazine's Game of the Year and Audio Design awards.

2.2 An Analysis of Video Game Music

Current Trends and Issues

It is clear to see from the above timeline that sound in video games has evolved rapidly, from the sonar-blip sound effect of *Pong* to specially commissioned orchestral soundtracks in under 30 years. However, it is only recently that attention has been paid to the music, as opposed to simple sound effects. Even now, the vast majority of games use music either incidentally or as ambient, looping tracks which play constantly, changing only when the action requires it. One may imagine that the latter situation is a step forward, but further examination shows that the implementation leaves much to be desired. A common gameplay scenario is used as an illustration:

1. The game is playing an arbitrary piece of background music, appropriate to the situation: for instance, the player is sneaking around an abandoned building, and there is 'tension' music playing.
2. An 'interesting' event happens in the game world: in our scenario, the player is spotted by an enemy.
3. This event sets off an in-game trigger to switch from the current music track to the 'action' music track. This is done with a simple crossfade.
4. Seconds later, the player kills the enemy; another trigger fires, and the music crossfades back to the 'tension' track, returning to the beginning of the piece.

The end result of this is a sudden, jarring transition to a different track which is musically very different – and then back again – over the course of just a few seconds. Repeated many times over the course of a game (or even a level), this rapidly becomes incredibly irritating to the player and can ruin the game experience.

How can this be resolved? Where can we look for inspiration and guidance? The most obvious of the modern media we can turn to is film, an industry with nearly a century's head start on games[3]. Parallels can be drawn between the history of cinema and the history of games, especially with respect to music: initially entirely a visual medium with no sound at all; then with simple background music played by onstage musicians; eventually, the problem of synchronised sound was overcome, resulting in the 'talkies' and culminating in the fully scored films of today.

[3]*Pong* was released in 1972, whereas the zoetrope was demonstrated in the 1860s and the first motion picture cameras in the 1880s. *Roundhay Garden Scene*, widely regarded as the earliest surviving motion picture, was filmed in October 1888.

Films, however, have one significant advantage over games: they are a passive medium, where the exact plot in all its detail is known in advance. Subsequently, it is possible for film composers to write the music in perfect timing to the action on-screen, so that the hero's punch is accompanied by a cymbal clash from the orchestra as well as a dramatic camera cut. This degree of synchronicity between action and music gives film the edge in creating, maintaining, and manipulating the mood of its audience; despite its best efforts, the gaming industry has still yet to come close to film's unparallelled ability to generate an emotional response to its characters.

Much talk is made of games attempting to emulate films, and much has been done in pursuit of this aim. Certainly visual realism has increased to the point where game visuals will be photorealistic within a decade – but in the area of music, games still trail film. What are the reasons for this?

- Firstly, it is a difficult problem. Games are an active medium: at any point the player usually has a vast number of options, and can choose to pick one at any time. This makes it impossible to write well-timed music in advance. Even with some kind of magical prediction algorithm to tell you exactly when the player is most likely to choose a certain action, the game would still require that the music 'writes itself' in real time.
- Secondly, music is a subtle medium, easily overshadowed by glitzy graphics or clever gameplay. Sound and music are far less obvious, and when their presence is noted, gamers tend to focus on sound effects rather than music – it is far more important that your gun 'sounds right' than it is for the music to be particularly well timed.

These are of course sweeping generalisations, and as with all generalisations, exceptions do exist. The games below use music as a core part of their gameplay, bringing the player's attention to it, rather than simply accompanying the action.

Dance Dance Revolution *DDR* and its many spin-offs are dancing games. As the music plays, arrows move from the top of the screen to the bottom (see Figure 2.1). When an arrow reaches the bottom of the screen, the player must step on the corresponding arrow on the pressure-sensitive pad at his feet. Points are scored for accurate timing and 'combos' – multiple steps performed correctly and within the allotted timeframe.

Guitar Hero / Rock Band These games put the players in the role of musician: each is equipped with a guitar-shaped controller with five 'fret buttons' and a 'strum bar' to simulate playing a guitar. In the case of *Rock Band*, a drum set (complete with drumsticks) and microphone can also be used. As with *Dance Dance Revolution*, on-screen graphics indicate when the notes on each instrument are to be played. Failure to 'play' an instrument skilfully enough results in the guitar track cutting out and the on-screen audience booing. While these games immerse the player in the music, it is the music which dictates the player's actions – the gameplay is merely glorified button-pushing.

REZ *REZ* is a 'rail shooter' released in 2001 on the PlayStation 2 and Dreamcast. The player's avatar moves on a predetermined path through the game world (the 'rail'), aiming and

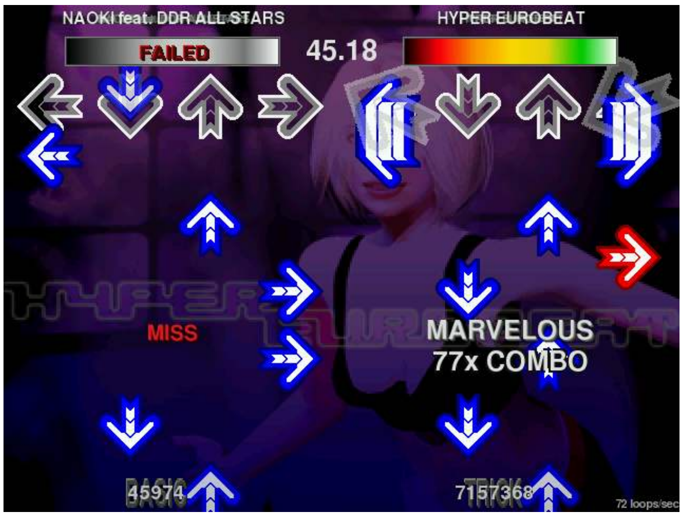


Figure 2.1: A screenshot of *Dance Dance Revolution*.

> firing to shoot down enemies as they are approached. In each level, a basic trance music track plays and is embellished by trills and extra drum beats, which are generated by the player's actions. This provides an extra level of immersion and the ability to adapt the music was marketed heavily as one of the game synesthetic qualities. It is important to note though that the player's actions are locked to the beat of the music, so a shot may be delayed slightly for the sake of keeping to the rhythm.

Of the above games, it is clear that *REZ* comes the closest to providing an adaptive musical experience during gameplay; the others employ music as a defining concept of the game, but they could easily be played with the sound off without losing much in the way of feedback to the player. Otherwise, there has been little development of audio engines in terms of adaptive music, with one exception: iMUSE.

iMUSE [27] is an audio system developed by LucasArts in 1991 to manage smooth audio transitions in their adventure games. Born from a composer's frustrating experiences with *The Secret of Monkey Island* [31], iMUSE allowed a composer to make multiple variations of the same piece of music (using different instruments and secondary themes, but with identical chord progressions), and set control points in these tracks. In-game triggers would alert iMUSE to switch to a different track at the next available control point. The system would then begin playing the second track, but starting at the point the previous track was at when the switch occurred. As all the tracks were synchronised and always in harmony with each other, this provided a smooth musical transition, allowing for subtly varying instrumentation and themes but still maintaining the overall structure and flow of the piece. The iMUSE system was first used in *Monkey Island 2: LeChuck's Revenge [32],* and was used in all subsequent LucasArts adventure games until the company underwent a restructuring in order to focus on *Indiana Jones* and *Star Wars* titles.

The Future – Games As Art?

Having looked at a timeline of music in video games and examined the more musically interesting games in more detail, we must ask the obvious question: what needs to change, and why? It has been noted that the film industry is ahead of the games industry with respect to creating music which matches the on-screen situation, but what motivation do games have to attempt to catch up, aside from avoiding the jarring musical transitions described earlier in this section?

The industry press rages with the ‘games as art’ debate (for an example, see [8]). Two schools of thought exist: those who believe that games are an art form, akin to sculpture, painting and music; and those who believe that games are a form of entertainment, more like the latest Hollywood blockbuster than a great work of literature. There are many arguments to be made for both sides, and a detailed examination of the debate is well beyond the scope of this report; however, one of the pivotal points in the discussion is that of emotion.

One of the arguments against the ‘games are art’ camp is that games fail to induce emotions in the player. While this is not strictly true – games can often be frustrating, for instance, as well as exciting and even exhilarating – they have so far lacked the ability to make the player experience a wider range of emotions: jealousy, anger, or even love. The current trend in games development is to attempt this, and some progress has been made: 2K Boston / 2K Australia’s *BioShock* [1] offers the player the moral choice of rescuing genetically modified young girls, or killing them for a greater reward; Lionhead Studios’ upcoming *Fable 2* [28] lets the player play a heroic adventurer, whilst caring for a family and an ever-faithful dog; and *Phoenix Wright: Ace Attorney – Justice for All* [6] puts the player in the position of a lawyer forced to defend a client who he believes to be guilty of murder. The emphasis, however, seems to be placed on good storytelling and clever character development, once again neglecting the subtle effect that music can have on mood. Combined with an absorbing storyline and the ability for the player to form ‘relationships’ with in-game characters, a suitably poignant refrain occurring at the right moment could be the final piece in the puzzle that moves games from entertainment to art.

Even aside from the ‘games as art’ debate and the quest to deliver emotional experiences to the player, we can still improve the gameplay experience by finding ways to avoid the jarring musical transitions mentioned at the beginning of this section, and generally to provide background music which is appropriate to the player’s state within the game world.

2.3 Previous Research in Emotion and Music

Much research has been done in the field of music psychology, and there has been some work done to apply this knowledge to the field of computer-controlled music. This section will summarise the research so far and suggest how it may be used in this project.

Basic emotions

The list of 'basic emotions' has been debated by theorists for many years: James [22] suggests a list of *fear, grief, love* and *rage*; while Plutchik [43] proposes a more expanded list comprising of *acceptance, anger, anticipation, disgust, joy, fear, sadness* and *surprise*. These lists the varying forms proposed by various theorists have long been used in the study of human emotional expression, as they are very easy to recognise and imitate. In their work at the University of Uppsala, Gabrielsson [14, 15] and Juslin [23, 24] suggest the list *anger, fear, happiness, sadness, solemnity and tenderness* and suggest that it is possible to map certain combinations of musical devices (tempo, dynamics, articulation, etc.) to emotions, so that applying these musical devices to an input piece would make it more likely to be recognised as the given emotion (see Table 2.1).

Bresin and Friberg [5] experimented with these cues in the program Director Musices (DM), creating various versions of two short pieces in different emotions, and then asking listeners if they could identify the emotion being expressed in the piece. Their results showed that translating the cues into direct modifications of the input piece resulted in listeners correctly identifying the perceived emotion of the modified piece in most cases.

Perceived and induced emotions

It is important to make the distinction at this point between *perceived* and *induced* emotion. Livingstone et al. [30] define perceived emotion as "the act of sensing the emotional content of the stimuli, or an emotional concept projected", whereas induced emotion is "that felt by the receiver after being subjected to the stimuli". For instance, a picture of a frowning face may be *perceived* by the viewer as angry, but whether the same picture *induces* anger is an altogether different question. This distinction is important as it has implications in assessing the effect of a piece of music on a listener: measuring perceived emotion is simply a case of asking the listener which emotion they think the piece is trying to express, whereas measuring induced emotion accurately is a far more difficult process. Livingstone et al. continue by suggesting that the induced emotion is dependant on three emotive response mechanisms: the threat assessment reactionary mechanism, or *fight-or-flight* principle; the empathic response mechanism; and prior emotional conditioning. These are difficult to identify, let alone control, and attempts to measure induced emotion through physiological means are further hindered by experimental inability to detect the valence (pleasantness) of an emotion, only the level of arousal. Even if measuring induced emotion accurately was possible, the actual emotion induced from a single piece of media may vary wildly over time, even with a single test subject. For these reasons, the majority of research on music-emotion correlations in music psychology has been based on measuring perceived emotions rather than induced ones. We shall adopt this viewpoint in order to harness the wider body of previous work done in the field.

Emotion	*Expressive Cue*	*Gabrielsson and Juslin 'cue profile'*
Fear	*Tempo*	Irregular
	Sound Level	Low
	Articulation	Mostly staccato or non-legato
	Time Deviations	Large Structural reorganisations Final acceleration (sometimes)
Anger	*Tempo*	Very rapid
	Sound Level	Loud
	Articulation	Mostly non-legato
	Time Deviations	Moderate Structural reorganisations Increased contrast between long and short notes
Happiness	*Tempo*	Fast
	Sound Level	Moderate or loud
	Articulation	Airy
	Time Deviations	Moderate
Sadness	*Tempo*	Slow
	Sound Level	Moderate or loud
	Articulation	Legato
	Time Deviations	Moderate
	Final Ritardando	Yes
Solemnity	*Tempo*	Slow or moderate
	Sound Level	Moderate or loud
	Articulation	Mostly legato
	Time Deviations	Relatively small
	Final Ritardando	Yes
Tenderness	*Tempo*	Slow
	Sound Level	Mostly low
	Articulation	Legato
	Time Deviations	Diminished contrast between long and short notes
	Final Ritardando	Yes

Table 2.1: Gabrielsson and Juslin's expressive 'cues', adapted from [5].

Hevner Adjective Circle and the 2DES

In 1935 Hevner published what is now widely referred to as the *Hevner adjective circle* [19] as a means of quantifying the perceived emotion of a piece of music. As shown in Figure 2.2, groups of similar adjectives are clustered together in a circular pattern. The groups are arranged so similar sets of emotions are near each other; as a result, each emotional transition from one group to the next is a fairly small one. This quantification of qualitative data simplifies data collection: listeners can simply pick the group that they think best describes the emotion the piece of music is trying to convey, and statistical analysis can be performed on the results.

Hevner's list of adjectives has been modified over the years: Farnsworth rearranged it into ten clusters and avoided the circular form [12], believing that it would continue to be modified [13].

In either form, the list has become widely used and can easily be superimposed onto the *two-dimensional emotion space* graph, as shown in Figure 2.2.

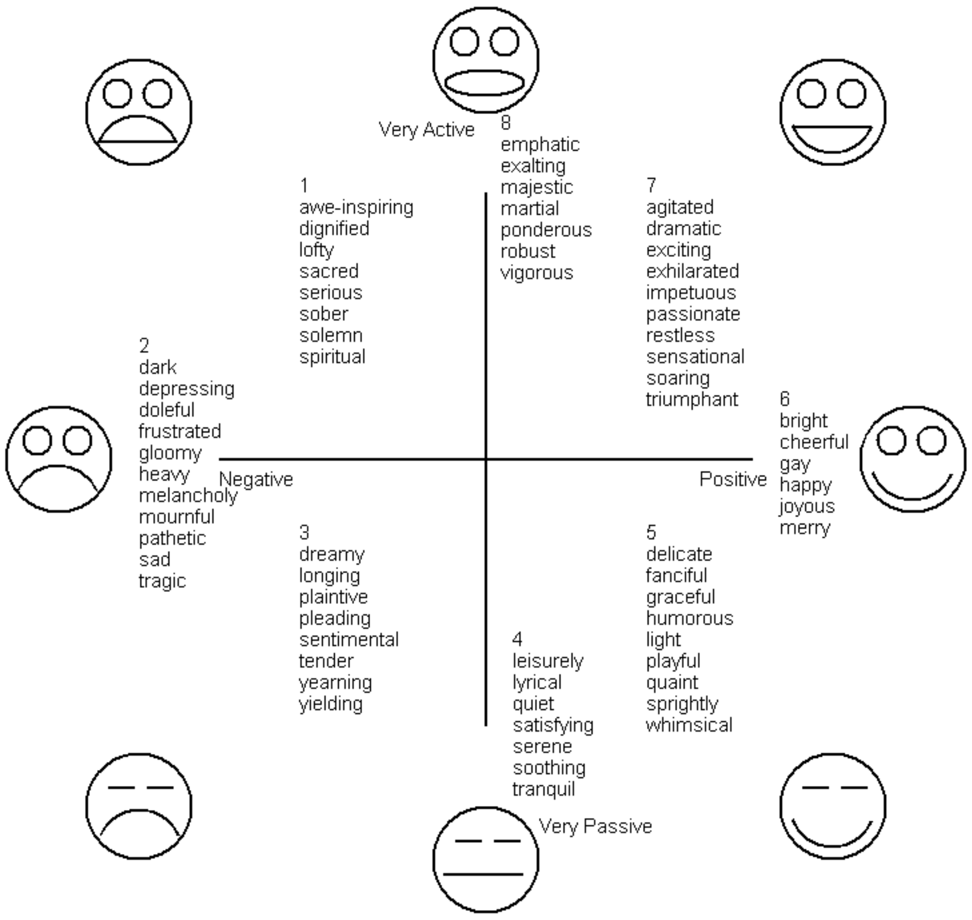


Figure 2.2: Hevner's original circle of adjectives superimposed onto the 2DES. Adapted from [46, 29].

The two-dimensional emotion space graph (henceforth referred to as the 2DES) proposed by Schubert [45] plots emotion states onto two axes: *valence* (pleasantness) and *arousal* (excitement). This allows for a continuous capture of emotional response by e.g. asking participants to draw a line on the graph indicating the current emotion as it changes throughout the piece.[4] Schubert followed up his work on the 2DES by empirically determining positions for adjectives in the Hevner circle on the graph [46]: for instance, Hevner group 7 (agitated, dramatic, exciting...) has positive values for both valence and arousal.

[4]It has however been noted that two-dimensional representations have trouble distinguishing between 'fear' and 'anger', and that a three-dimensional approach (with *dominance/submission* as the third axis) would separate the two. However, it would only provide a marginal improvement while increasing the attentional load on the participant while conducting the investigation. [54]

In 2005, Livingstone and Brown [29] collated the results from various studies and published a table mapping octants of the 2DES to musical rules such as mode and articulation. Analysis of this table shows that there are marked correlations between musical properties and the 2DES axes: fast tempo indicates high arousal, a minor mode with complex harmony indicates a negative valence, and staccato articulation indicates high arousal and positive valence. The high degree of correlation between musical properties and perceived emotions further suggests that appropriate use of musical devices and properties can push the perceived emotion of a piece in a particular direction. Indeed, the same study took two pieces of music – a Mozart string quartet and a fragment of music from *The Legend of Zelda: Ocarina of Time* – and applied the musical rules consistent with an octant of the 2DES. Testing was reasonably successful, showing that application of the correct rules could influence the perceived emotion of the piece in a particular direction. There were some issues with the results: as both of the original pieces were upbeat and happy, listeners tended to continue choosing high valence values for the modified pieces.

Summary

While the use of music in games is becoming more prevalent, games are still failing to tightly integrate their music tracks with the gameplay. The wide body of research into music psychology provides ample information on the links between music and emotion, and continues to be an active area of research. Harnessing this research and using the results effectively in games could lead to an increased emotional connection between the player and the game world. In the next section, we propose a way of using such research to improve background music in games.

Chapter 3

Adaptive Music

Chapter 2 showed us how games are failing to provide a seamless musical experience, often naïvely switching between distinct tracks, leading to jarring transitions between them. This section proposes a method of adapting the music to the gameplay situation in a less noticeable manner which should be more pleasing to the player.

3.1 The Problem

For ease of reference, we restate the problem. It is fairly simple:

> **Modern games use a variety of distinct music tracks and change between them at trigger points during the course of the game. The change is handled using a simple crossfade, which can provide a very jarring transition as the tracks are often musically very different. This is unpleasant and distracting for the player.**

While there are games which do not suffer from this problem, and even technologies (such as iMuse, detailed on page 17) designed to avoid this, the industry in general has failed to address the issue.

3.2 The Proposed Solution

The proposed solution is equally simple:

> **Instead of using multiple tracks for different gameplay situations, use a *single* music track throughout the course of the game, and modify it as it plays to suit the situation.**

This way, the musical shape and theme of the piece will be retained, minimising the effect of the transition while providing different musical attributes for different gameplay situations.

3.3 Requirements

In order to create a system that is able to adapt the notes and attributes of the background music, the music itself needs to be represented in a format which is easy to parse and alter in real-time. For this reason, MIDI has been chosen as the input filetype: details of the MIDI specification are provided in Appendix B. Suffice it to say that the MIDI specification is widely understood and documented, so parsing and altering it should not be too problematic.

Such a real-time MIDI adaptation system should provide suitable adjustments necessary to shift the perceived emotion of the piece in a given direction. This must happen in real-time, without gaps in the audio playback, if the system is going to be used within the environment of a computer game.

To summarise, the system should be able to:

1. Adapt an entire MIDI file to shift its perceived emotion in a given direction by a set amount:
    (a) This requires some knowledge of the links between the musical attributes the system is able to modify, and the perceived emotions of the piece after such modifications are applied. These links should then be distilled into rules that the system can apply in order to create an adaptation of the piece with a given emotion.
    (b) The system should cater for a wide range of emotions: a system that can only produce 'angry' pieces will not be able to produce an appropriate range of music tracks for use in varying gameplay scenarios.
    (c) The degree of perceived emotion achieved should be adjustable: that is, for a given piece it should be possible to create at least two distinct 'angry' tracks, where one sounds angrier than the other, but where both sound angrier than the input track.
    (d) The system should produce pieces that have the same perceived emotion as the intended emotion: that is to say, pieces that are meant to sound angry should sound angry rather than happy or sad.
2. Play back a MIDI file and 'push around' its perceived emotion in real-time:
    (a) The system should be able to cope with applying the rules for each emotion in real-time, *and also leave enough CPU time for the game to run on.* It is not sufficient for the system to be able to run well on its own if it negatively affects the framerate of the game it is coupled to.

(b) The system should be able to handle changing emotions gracefully. This may mean that it is possible to specify a time interval over which the emotion is required to change, or that a series of small changes can be strung together over time to make a large, gradual change. In either case it should be possible to change emotions both suddenly and gradually without the musical transition being too jarring.

Such a system would then be suitable for applying to a computer game, using in-game triggers to start the real-time adaptation of the input piece as it is being played. This would, then, be able to achieve the goal of smooth musical changes that were still capable of reflecting the game situation. In order to design this system, we must first gather a set of music-emotion rules, such as those referred to in 1a above. For this, we need to conduct a more careful examination of the research outlined in Section 2.3.

Chapter 4

Transforming Music

4.1 Introduction

The works cited in Section 2.3, particularly that of Livingstone and Brown [29], show that it is indeed possible to computationally alter the perceived emotion of a piece of music by applying well-defined rules. Whether this translates to moving around the 2DES or moving between the basic emotions, it certainly seems that applying quantitative changes to a work can influence the listener's opinion of what the piece is attempting to express. It would follow, then, that applying these rules as a piece is playing would alter the perception of the piece's emotion over time. Applying these principles to game design suggests that it is indeed possible to adjust the currently playing music track to suit the in-game situation, rather than abruptly switch to a new one: the game engine should be able to 'work out' what sort of situation the player is currently in (safe or dangerous, for example) and choose an appropriate emotion; from here, modifying the currently playing track in that direction has been shown to be possible. This chapter builds on the work of Livingstone and Brown to suggest a number of musical 'transformations', each of which will change the perceived mood of a given piece of music. We then analyse how these transformations could be applied to a MIDI file, with the intent to play such a file as the background music for a computer game, and dynamically alter the music at various trigger points in the game corresponding to an appropriate change of mood.

2DES Literature Review

The literature review carried out by Livingstone and Brown provides a wealth of information, correlating various musical properties with octants of the 2DES (Table 4.1). The main results of the review are summarised graphically in Figure 4.1 and Figure 4.2. From Figure 4.2, we can see that the top five musical transformations cover 80% of all the recorded correlations; it is these that we shall examine more carefully in this chapter.

Octant	2DES Octant Rules
1	Mode Major(19), Tempo Fast(16), Harmony Simple(8), Loudness Loud(7), Articulation Staccato(5), Pitch High(3), Rhythm Flowing(3), Pitch Range High(2), Pitch Variation Large(2), Pitch Contour Up(2), Note Onset Rapid(2), Rhythm Smooth(2), Rhythm Activity(2), Loudness Medium(1), Loudness Soft(1), Loudness Variation Small(1), Loudness Variation Rapid(1), Loudness Variation Few(1), Pitch Low(1), Pitch Range Low(1), Pitch Contour Down(1), Timbre Few(1), Rhythm Rough(1)
2	Tempo Fast(20), Loudness Loud(10), Mode Major(8), Pitch High(4), Pitch Variation Large(4), Harmony Simple(4), Note Onset Rapid(4), Pitch Range High(3), Pitch Contour Up(3), Articulation Staccato(3), Articulation non-legato(2), Harmony Complex(2), Rhythm Flowing(2), Rhythm Activity(2), Rhythm Smooth(2), Loudness Variation Small(1), Loudness Variation Few(1), Loudness Variation Rapid(1), Pitch Low(1), Pitch Range Low(1), Pitch Variation Small(1), Pitch Contour Down(1), Timbre Few(1), Timbre Many(1), Tempo Slow(1), Vibrato fast(1), Rhythm Complex(1), Rhythm Firm(1), Metre Triple(1), Tonality Tonal(1)
3	Mode Minor(14), Loudness Loud(9), Tempo Fast(9), Harmony Complex(8), Note Onset Rapid(5), Pitch Contour Up(5), Pitch High(4), Pitch Range High(3), Pitch Variation Large(3), Loudness Soft(2), Rhythm Complex(2), Loudness Variation Large(2), Timbre Sharp(2), Articulation Non-legato(2), Pitch Variation Small(2), Articulation Staccato(2), Note Onset Slow(2), Timbre Many(1), Vibrato Fast(1), Rhythm Rough(1), Metre Triple(1), Tonality Tonal(1), Tonality Atonal(1), Tonality Chromatic(1), Loudness Variation Rapid(1), Pitch Low(1)
4	Mode Minor(12), Harmony Complex(6), Articulation Legato(3), Pitch Variation Small(3), Tempo Fast(3), Loudness Loud(2), Loudness Soft(2), Loudness Variation Large(2), Note Onset Rapid(2), Note Onset Sharp(2), Note Onset Slow(2), Timbre Sharp(2), Loudness Variation Rapid(1), Pitch High(1), Pitch Low(1), Pitch Range High(1), Pitch Variation Large(1), Pitch Contour Up(1), Pitch Contour Down(1), Timbre Many(1), Harmony Melodic(1), Tempo Slow(1), Articulation Staccato(1), Rhythm Complex(1), Tonality Atonal(1), Tonality Chromatic(1)
5	Tempo Slow(15), Articulation Legato(6), Mode Minor(7), Harmony Complex(7), Loudness Soft(3), Harmony Simple(3), Pitch Low(3), Note Onset Slow(3), Pitch Range Low(2), Pitch Contour Down(2), Rhythm Firm(2), Loudness Loud(1), Loudness Variation Small(1), Loudness Variation Few(1), Pitch Variation Small(1), Pitch Contour Up(1), Mode Major(1), Timbre Few(1), Timbre Soft(1), Harmony Melodic(1), Note Onset Rapid(1), Vibrato Deep(1), Rhythm Smooth(1), Tonality Chromatic(1)

6	Loudness Soft(5), Tempo Slow(5), Pitch Variation Small(3), Articulation Legato(3), Note Onset Slow(3), Pitch Low(3), Pitch Range Low(2), Loudness Variation Rapid(1), Pitch High(1), Pitch Contour Down(1), Mode Minor(1), Timbre Few(1), Harmony Complex(1), Vibrato Deep(1), Metre Duple(1), Tonality Tonal(1)
7	Tempo Slow(10), Loudness Soft(9), Articulation Legato(5), Note Onset Slow(3), Pitch Low(2), Pitch Range Low(2), Pitch Variation Small(2), Timbre Soft(2), Harmony Simple(2), Mode Minor(1), Loudness Variation Rapid(1), Loudness Variation Few(1), Pitch High(1), Note Onset Rapid(1), Vibrato Intense(1), Rhythm Smooth(1), Rhythm Flowing(1), Rhythm Firm(1), Metre Duple(1)
8	Mode Major(12), Harmony Simple(5), Tempo Slow(3), Articulation Staccato(3), Loudness Soft(2), Pitch Contour Up(2), Loudness Variation Few(1), Pitch High(2), Pitch Range High(1), Pitch Range Low(1), Pitch Low(1), Mode Minor(1), Timbre Soft(1), Vibrato Intense(1), Rhythm Smooth(1), Rhythm Flowing(1), Tonality Tonal(1)

Table 4.1: Comparative 2DES Literature Review, reprinted with permission from [29]. Numbers in (brackets) indicate the number of studies finding a correlation between the given property and octant.

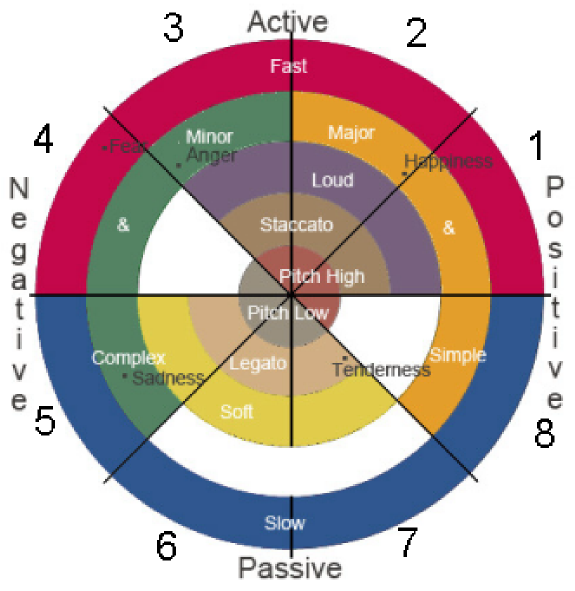


Figure 4.1: The primary music-emotion structural rules, with their areas of effect shown on a 2DES graph with octants labelled as in Table 4.1. Reprinted with permission from [30].

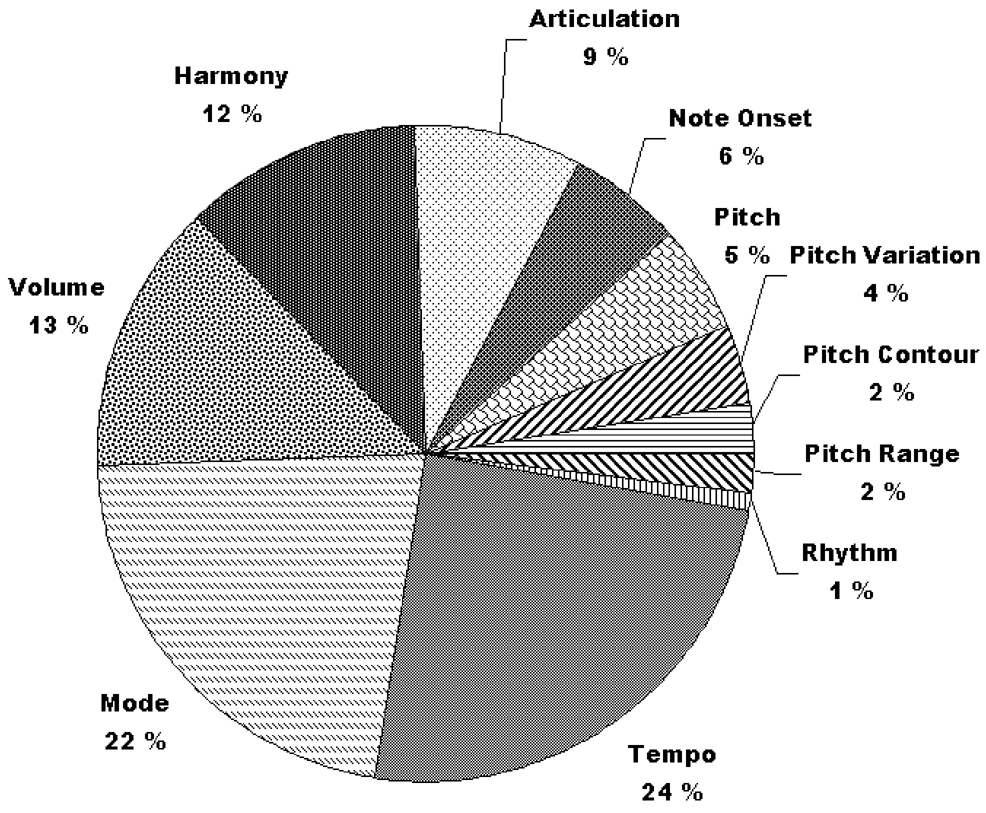


Figure 4.2: Pie chart showing comparative incidences of musical properties reviewed in Table 4.1. Note that the five most commonly occurring properties account for 80% of the observations.

4.2 Transformations

4.2.1 Mode

Figure 4.1 shows us that the area of effect of changing mode between major and minor is mirrored around the arousal axis of the 2DES. Specifically, a major key is typical for octants 1, 2, and 8, and a minor key is typical for octants 3, 4, and 5. Transforming a piece from a given key to its parallel major or minor is a fairly simple task, requiring transformation of only the third, sixth, and seventh intervals of the scale (see Figure 4.3). Extra care should be taken when transforming to a minor scale to ensure that the seventh is raised in an upward contour, and lowered in a downward contour.

Transforming a piece to its relative major or minor would take significantly more work, as the melody line would need to be rewritten using a scale that started from a different tonic to the original. Additionally, the correct chord progressions would need to be used in order to maintain a smooth melodic change to the new key, and doing so would be difficult as the transformation could be required to start at any time; this would mean creating a chord progression from an artibrary chord.

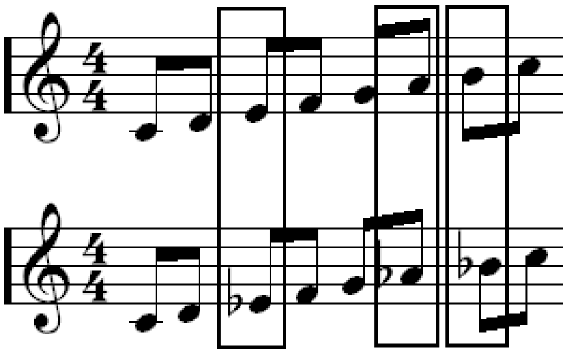


Figure 4.3: Transformation from C Major scale (top) to C Minor scale (bottom, technically the *natural minor* scale - see Appendix A.2). The notes that need to be transformed are III, VI and VII, highlighted.

4.2.2 Tempo and Articulation

The correlation between tempo and mood is obvious to even the casual listener: faster music indicates more arousal. However, tempo fluctuations such as those found in rubato can also act as an emotional intensifier. Figure 4.1 shows us that the area of effect of changing articulation is roughly mirrored around the valency axis of the 2DES: staccato articulation is more typical of aroused music than legato articulation. In computer generated music, articulation can be simulated by lengthening or shortening the amount of time the notes are held down.

4.2.3 Volume

Figure 4.1 shows us that the area of effect of changing volume is roughly mirrored around the valency axis of the 2DES. Louder music is often seen to be more positive and active; quieter music is more relaxed and negative. However, the subtlety in this is shown well in the graphic: loud music can have a negative valency (anger), and quiet music can be positive (tenderness).

4.2.4 Harmony

Figure 4.1 shows us that the area of effect of changing articulation is mirrored around the arousal axis of the 2DES. More complex harmony is associated with negative valency, and simpler harmony is associated with positive valency. However, it is unclear what exactly is meant by ‘complex’ or ‘simple’ harmony, although it could be reasonably assumed that it refers to the presence of auxiliary notes in chords. In Livingstone and Brown’s [29] implementation, a harmony simplification rule was used, to remove notes from large chords.

Summary

It is clear from the work in this chapter that each musical transformation has a specific area of influence on the piece’s perceived emotion. This is a pleasing result, as it paves the way for the process of applying each transformation to varying degrees, eventually arriving at a target emotion suitable for the current gameplay situation.

Chapter 5

ACRONYM

ACRONYM consists of five 'transformation' classes and a controller class to iterate over the MIDI files and call the transformations as necessary. As detailed in Appendix C.2, the jMusic library functions `Read.midi()` and `Write.midi()` were modified to take into account key and time signatures and import them into intermediate data structures for later use.

This chapter will detail the overall program flow of ACRONYM, and the responsibilities of the main parts of the program.

5.1 Program Flow

Figure 5.1 indicates the program flow of ACRONYM. The system is triggered by an external event calling `Adaptive.init()` (see §5.2 for details on individual classes in ACRONYM), which reads in the chosen MIDI file and converts it to Format 0 if necessary (more details on the MIDI specification can be found in Appendix B). It then waits for `Adaptive.play()` to be called, which begins playback of the file. Once `Adaptive.adjust()` is called, the static variables in the *Articulator*, *Volumiser*, *Transposer* and *Harmoniser* classes are set up and execution passes to the *Adapter*.

The *Adapter* then informs the *SeqSynth* class that the sequence is about to change and the original sequence should be loaded. It then makes a copy of the event list and loops through all events, updating the *SeqSynth*'s sequence with the altered event after calling the various transformation classes. At the end of each beat, the *Harmoniser* is called to add or remove notes from any chords which began in that beat.

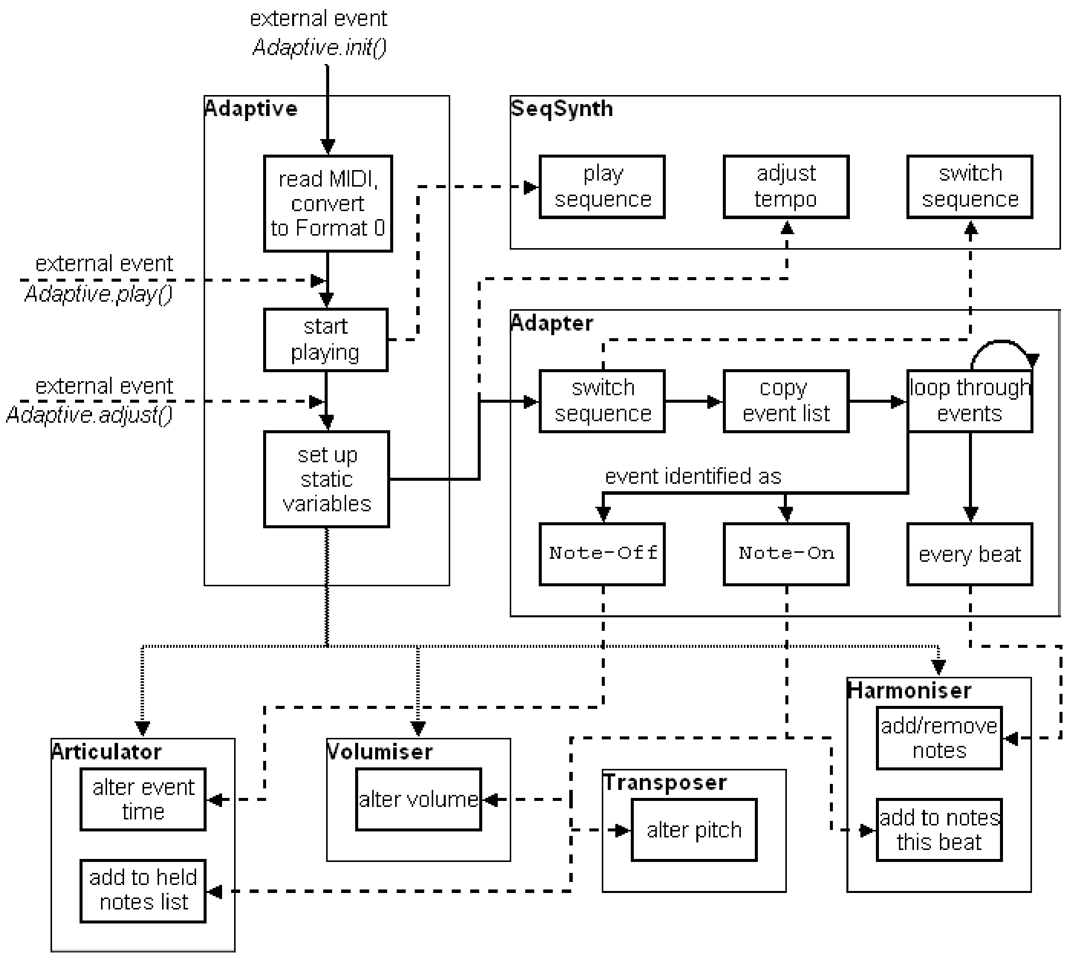


Figure 5.1: ACRONYM program flow. Solid lines indicate program flow, fine dotted lines are initialisation, and dashed lines are inter-class communication. Thin boxes represent class boundaries, thick boxes represent class actions.

5.2 Program Classes

5.2.1 Adaptive

Adaptive is the main class of the program, functioning both as an entry point and providing an API for developers wishing to use ACRONYM's real time music adaptation features. It provides the following functions:

`init( String` ***filename*** `)` Loads in the MIDI file *filename*, converting it to the jMusic intermediate format (see Appendix C) and also into a Format 0 SMF if necessary. Creates a new *SeqSynth* (§5.2.2), as well as new *Adapter* (§5.2.3) and *TempoScaler* (§5.2.5) threads.

`play()` Starts playing the loaded MIDI file.

`stop()` Stops playing the loaded MIDI file.

`adjust(`***mode*, *tempoFactor*, *articulation*, *volume*, *harmony*, *time***`)` Adjusts the currently playing MIDI file over *time* seconds to a different *mode* (0 = major, 1 = minor), scaling the original tempo by a factor of *tempoFactor*, note lengths by *articulation*, note volumes by *volume*, and adding or removing *harmony* percent of the notes. Interrupts the execution of any currently running *Adapter* thread and starts a new one after setting up the transformation classes with the given parameters.

`adjust(`***x*, *y*, *time***`)` Adjusts the currently playing MIDI file so the music moves smoothly to position (*x*, *y*) on the 2DES over *time* seconds. Converts the 2DES coordinates and then calls the `adjust()` function above.

ACRONYM does not do key detection, and key signature events in MIDI files are often incorrect, most often indicating the key signature of the relative major for minor pieces. To combat this, the `init()` function also checks the file `minor_override.txt` to see if the *filename* is listed. If it is, ACRONYM will automatically assume all key signature events refer to the relative minor of the key given. This gives the user the option to specify some files as being incorrect, rather than forcing them to use a different file.

5.2.2 SeqSynth

SeqSynth is ACRONYM's sequencer and synthesizer class. It attempts to find the Java *Real Time Sequencer* class, the only Java sequencer which allows the MIDI sequence to be modified while it is playing. It also looks for an appropriate MIDI synthesizer on startup, as well as managing all communication between the system and the sequencer.

5.2.3 Adapter

The *Adapter* is the main class which makes the necessary alterations to the MIDI file in real time. A new *Adapter* thread is spawned (and any old threads killed) each time `Adaptive.adjust()` is called; this thread then runs independently of the playing MIDI file, calling the transformation classes as necessary. As repeated mode changes do not preserve the original piece (see §5.2.4), the *Adapter* modifies a fresh copy of the original MIDI file each time `Adaptive.adjust()` is called.

Switching MIDI sequences

There were two possible approaches to handling the real-time transformation of the input MIDI file:

1. Use the full abilities of the Java *Real Time Sequencer* and modify the currently playing sequence on-the-fly. This could cause problems as there would be no way of knowing how far previous *Adapter* runs had made it through the file before being terminated, meaning that the input for the new *Adapter* process would be a partially-altered MIDI file. The input would therefore be unpredictable, with varying amounts of transformations applied through the piece, resulting in a disjointed overall effect as the system would not be able to tell which parts of the piece had previously been transformed.

2. Modify a fresh copy of the original MIDI file and switch the currently playing MIDI file to the new one. This method would ensure there was always a reliable, unmodified version of the MIDI file to transform, at the expense of a slight hitch in sound when the switch occurred.

I took the second option as I felt the need for a smoother, more reliable transformation greatly outweighed the small hitch in sound.

Execution

When a new *Adapter* thread is created by *Adaptive*, the `run()` method is automatically called. The `run()` method tells *SeqSynth* to switch to a fresh copy of the original MIDI file and continue playback from the same point, and then begins to scan through a *copy*[1] of the original MIDI file, one event at a time. Each event in the copy is looked up using a binary search in the MIDI file just sent to the sequencer and undergoes a series of transformations:

- If the event is a `Key Signature` or `Time Signature` event, the current key signature or time signature is updated.
- If the event is a `Note-On` or `Note-Off` and the event time is before the synthesizer's current position in the track, it is skipped.
- If the event is a `Note-On` and the event time is after the synthesizer's current position in the track, it is sent to the *Transposer* (§5.2.4) for rescaling, to the *Volumiser* (§5.2.7) for changing volume, and its details are added to the *Articulator* (§5.2.6) and *Harmoniser* (§5.2.8).
- If the event is a `Note-Off` and the event time is after the synthesizer's current position in the track, it is sent to the *Transposer* (§5.2.4) for rescaling, and the *Articulator* (§5.2.6) for changing position in the track.
- Finally, the *Harmoniser* (§5.2.8) is called at the end of every beat to add or remove notes within that beat.

[1]This is necessary as the *Articulator* (§5.2.6) transformation does not preserve the ordering of the events. Modifying the currently-playing file directly would mean some events would be transformed twice, and possibly others to be skipped entirely.

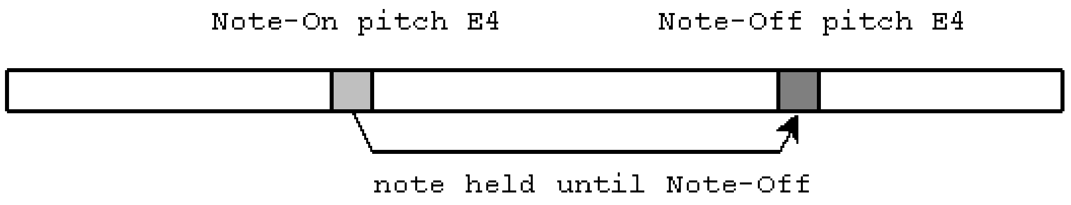
The new event, complete with transformations, is then returned to the currently playing MIDI file in the correct position.

5.2.4 Transposer

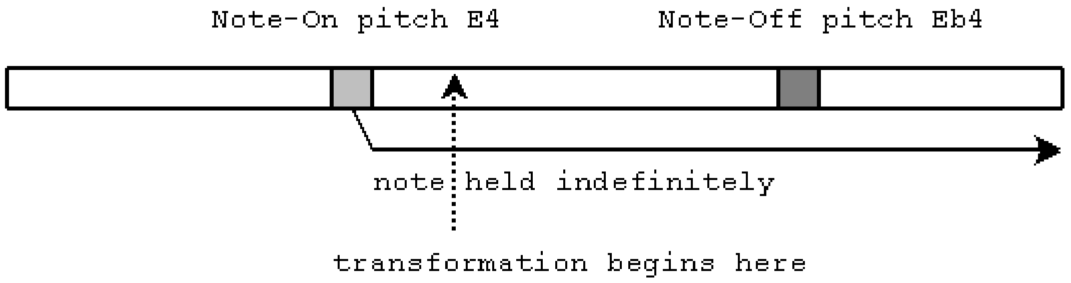
The *Transposer* uses an input scale, `inScale`, and an output scale, `outScale`, to move notes between the original key and its parallel major or minor. Each note is looked up in `inScale` and its position is noted; the note is then given the new pitch given in the position in `outScale`. If the note does not exist in `inScale` (perhaps because it is a bluesy seventh note), the closest note in `outScale` is used. If the note in question is a seventh, and `outScale` is a minor scale, the previous note on that channel is examined. If the previous note is higher than the current one, the minor seventh is used; if the previous note is lower, the major seventh is used.

The *Transposer* relies heavily on the use of `Key Signature` events in the MIDI file. If these are not present, ACRONYM assumes the key is C Major (as per the MIDI specification, see Appendix B); if this is not truly the case, the resulting transformation will be incorrect and sound rather out-of-key.

The 'Orphaned Note' Problem



(a) Initial layout of a MIDI file in the key of C Major, prior to transformation.



(b) MIDI file after transformation to C Minor at the point shown. Note that the `Note-On` event shown now has no corresponding `Note-Off` event, and will thus be held down indefinitely.

Figure 5.2: A demonstration of the 'orphaned note' problem.

During development I discovered that immediately after `Adaptive.adjust()` was called, some notes carried on sounding indefinitely. This turned out to be because some `Note-Off` events happening after the transformation call were being passed to the *Transposer* and changed pitch; this left their corresponding `Note-On` events held down (Figure 5.2). This problem was solved by introducing an *OrphanedNoteLogger* class to monitor all notes which are currently being 'held down' by the MIDI synthesizer. Any transformation for a `Note-Off` event within this list are skipped, allowing all previously held notes to be released.

5.2.5 TempoScaler

The *TempoScaler* is the simplest transformation class: it is simply a thread which repeatedly sends commands to the sequencer to alter the tempo scale factor over a period of time. This has the side-effect of being only a performative transformation: were the transformed MIDI sequences to be written out to a file, the tempo would remain as in the original, as the tempo events are not changed.

5.2.6 Articulator

The *Articulator* stores the timepoint of every `Note-`On event until its corresponding `Note-Off` event is reached. It then sets the timepoint of the `Note-Off` to be a factor of the current difference between the two event times:

```
Note-On t=1000  Note-Off t=1050
Articulator scale factor: 1.5  New Note-Off t=1075
```

This transformation class is the reason ACRONYM uses a copy of the original MIDI file and then looks up the same event in the currently playing sequence (also referred to in the footnote to §5.2.3). If the system modified the currently playing sequence directly and the *Articulator* scale factor was greater than 1, any `Note-Off` event would be replaced into the sequence ahead of its old position, and subsequently read and pushed forward repeatedly. The net effect of this would be to move all `Note-Off` events to the end of the piece.

5.2.7 Volumiser

The *Volumiser* scales the volume of all Note-On events by a given factor. It also contains functionality to emphasise the first and third beats in a $\frac{4}{4}$ time signature, or the first beat in any other time signature; however, this functionality was not used as there was no reliable data to suggest what effect this may have on perceived emotion.

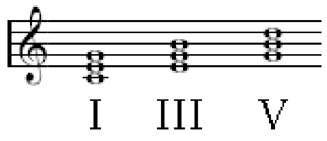
5.2.8 Harmoniser

Using the reasoning of Livingstone and Brown's [29] implementation, ACRONYM takes 'harmony' to mean 'texture'. However, ACRONYM provides methods to thin *and* thicken the texture of a piece. The system uses a *ChordAnalyser* class to analyse each chord and rate each note in the chord. As with the *Transposer*, the *Harmoniser* relies heavily on the use of `Key Signature` events in the MIDI file and will produce incorrect results if they are not present.

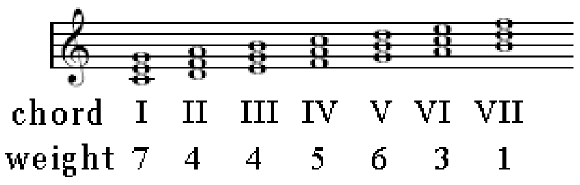
Chord Analysis

Each `Note-On` event is passed to the *ChordAnalyser*. If the note occurs within the first half of a beat, it is considered to be part of that beat's chord and is added to the *chord list* and the *note list* of chords occuring in that beat; otherwise it is just added to the *note list*. The chord for a given beat is determined by analysing all notes in the *chord list* and creating a list of *triad probabilities* as follows (Figure 5.3):

- Each note is located in the scale of the piece's current key. If the note is not in the scale, it is not used to determine the chord.
- If the note is in the scale, it is identified to be belonging to one of three chords, and the list of *triad probabilities* is updated appropriately (Figure 5.3a).
- Once all chords in the chord list have been examined, the *triad probabilities* are weighted to reflect the relative frequency of occurrence of the triads (Figure 5.3b). The triad with the highest final score (`triadProbabilities[i] * weight`) is then deemed to be the chord for that beat.



(a) The note G (second line from the bottom) in the key of C Major could belong to a chord starting on the tonic, mediant, or dominant. In this case, the count at `triadProbabilities[0]`, `triadProbabilities[2]`, and `triadProbabilities[4]` would be incremented.



(b) Relative weightings of triads in the key of C Major.

Figure 5.3: Determining the chord for a given beat.

Once the chord for the beat is determined, each note in that beat (i.e. each note in the *note list*) is given a rating as to how appropriate it is for that chord. Notes in a given chord are rated as follows:

- The tonic of the chord is given a rating of 8.
- The mediant of the chord is given a rating of 4.
- The dominant of the chord is given a rating of 6.
- The leading note of the chord is given a rating of 2.
- All other notes are given a rating of 0.

The *Harmoniser* then calls `thinTrack` or `thickenTrack` depending on whether the texture scale factor is below or above 1.

Thinning

The *note list* is sorted according to rating and the `Note`-On events for the lowest n notes are removed, where n = `scaleFactor * sizeof(noteList)`.

Thickening

New notes are created as follows:

- A `timepoint` for the new note is selected at random from the set of all timepoints of notes in the *note list*.
- A `channel` for the new note is selected at random from the set of all channels of notes in the *note list*.
- The `volume` for the new note is the average of all notes in the note's `channel` in *note list*.
- A `pitch` for the new note is selected. This pitch is either the tonic, mediant, dominant or leading note, with relative probability ratios 8:4:6:2.
- The `pitch` is transposed so it is within the average pitch range of the chosen `channel`.

A new `Note-On` event with the parameters `channel pitch volume` is inserted at `timepoint`, and a new `Note-Off` event with the parameters `channel pitch 64` is inserted at the end of the beat. This process is repeated as many times as is necessary to increase the number of notes in the beat by the scale factor.

Chapter 6

Use of ACRONYM and Demo Applications

This chapter details the intended use of ACRONYM in a commercial game, and outlines the function and purpose of the *Standalone Music Adaptation Tool* and *Gun Man: Struggle Eternal* demonstration applications.

6.1 Use of ACRONYM

ACRONYM is designed to be used primarily in a commercial computer game. Commercial computer games are generally written on top of a game engine, created either by the developers or licensed from another company. The purpose of the game engine is to abstract away the details of rendering, collision etc. and provide a high-level API for developers to work with. Most modern game engines contain at least the following functionality (Figure 6.1):

Graphics The graphics section of the engine is usually the most complex and will contain detailed instructions for the graphics processing unit (GPU) for rendering game world objects on screen. This allows game developers to import and manipulate 3D models directly, letting the graphics engine handle the lighting and drawing to screen.

Physics The physics engine handles collisions between game entities and resolves the forces in order to simulate the real world more accurately. These calculations are done on the CPU, or on a dedicated physics processing unit (PPU).

AI Occasionally, AI is handled by the game engine, providing a framework for agents to be developed on.

Networking For multiplayer games, game engines will often have a networking layer which deals with ensuring the game state is synchronised across all players.

Audio The audio part of the game engine is concerned with loading and playing sound files and applying effects to them. It may also provide trigger routines for changing background music tracks.

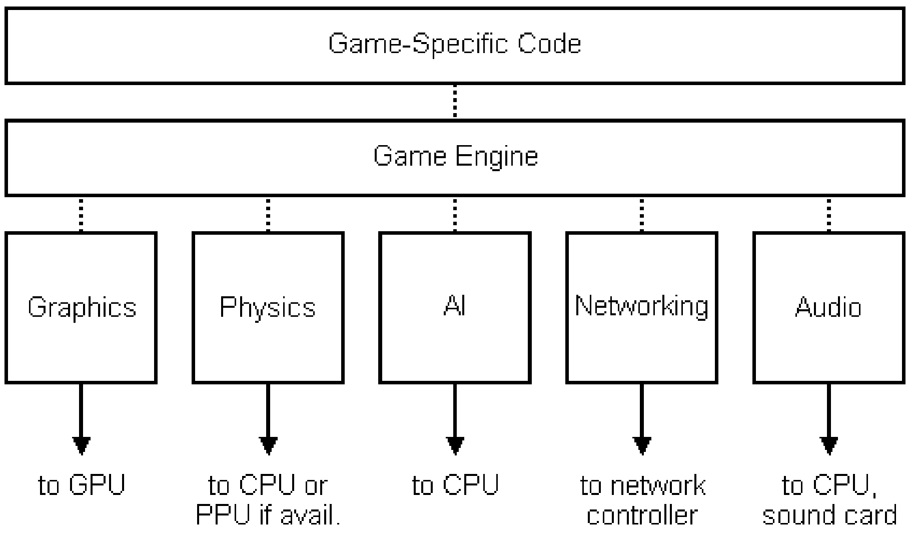


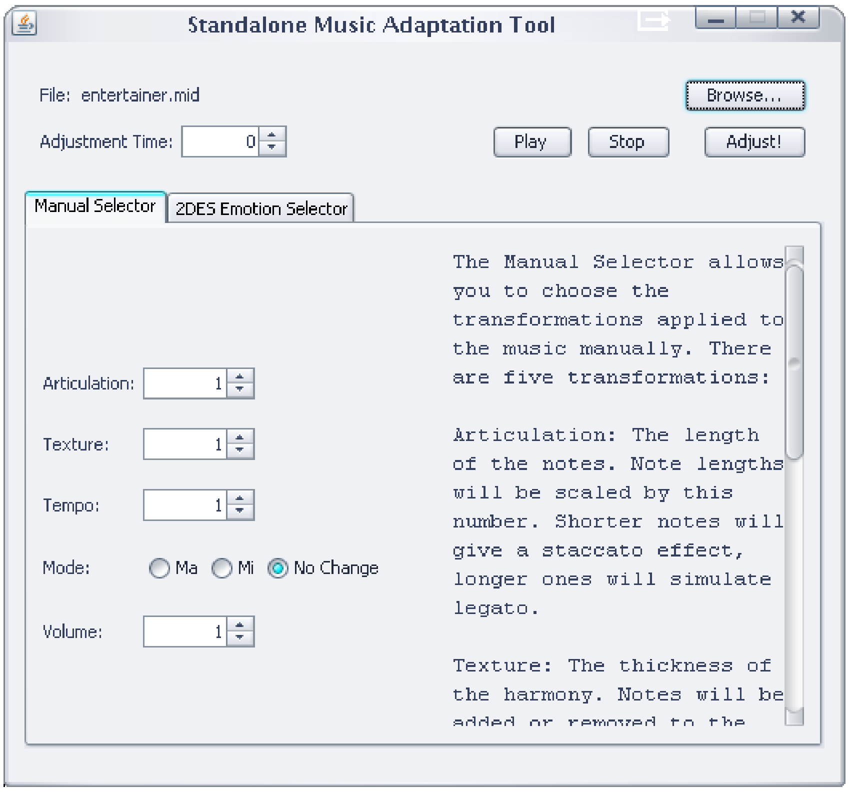
Figure 6.1: Breakdown of a typical game.

ACRONYM is designed to supplement the development of audio for a game; in that sense, it can be regarded as a supplemental audio engine. Its purpose is to give game developers a method of altering a single piece of music towards a certain emotion, without requiring an indepth knowledge of music theory. All that is required is an understanding of the 2DES, which is fairly simple to grasp. The API available to developers is detailed in Section 5.2.1.

6.2 Standalone Music Adaptation Tool

The Standalone Music Adaptation Tool (SMAT) allows the user to access all features of ACRONYM using a simple, graphical tool. It offers two ways to adjust a MIDI file: by altering the parameters directly, or by selecting a target emotion point on the 2DES and letting the system adapt the music according to its internal rules. The adjustment can then be applied instantly, or set to be applied gradually over a number of seconds.

6.2.1 Manual Selector



File: entertainer.mid

Adjustment Time: 0

Articulation: 1

Texture: 1

Tempo: 1

Mode: ( ) Ma ( ) Mi [✓ No Change]

Volume: 1

The Manual Selector allows you to choose the transformations applied to the music manually. There are five transformations:

Articulation: The length of the notes. Note lengths will be scaled by this number. Shorter notes will give a staccato effect, longer ones will simulate legato.

Texture: The thickness of the harmony. Notes will be added or removed to the

Figure 6.2: The SMAT Manual Selector.

The Manual Selector tab (Figure 6.2) allows the user to select precisely the adjustments desired and then passes the resulting values to `Adaptive.adjust()`. The values are as follows:

Articulation Scale factor for the articulation of the piece. Notes will be lengthened or shortened by this factor of their initial length.

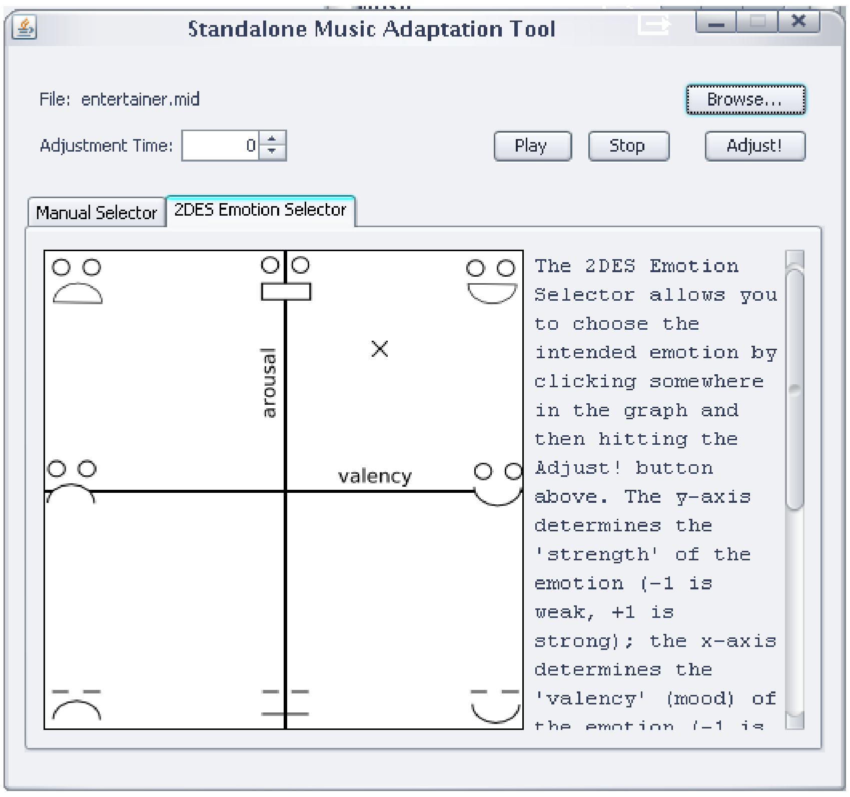
Texture Scale factor for the texture of the piece. Notes will be added or removed until the number of notes in the beat is *initial * texture*.

Tempo The tempo will be multiplied by this value number.

Mode Allows the user to change the piece's mode to major, minor, or to make no change. If the 'Ma' radio button is selected and the piece is already in a major key, no changes will be made to the mode; similarly, if the 'Mi' radio button is selected and the piece is already in a minor key, no changes will be made.

Volume The volume of all notes in the piece will be multiplied by this number.

6.2.2 2DES Emotion Selector



The 2DES Emotion Selector allows you to choose the intended emotion by clicking somewhere in the graph and then hitting the Adjust! button above. The y-axis determines the 'strength' of the emotion (-1 is weak, +1 is strong); the x-axis determines the 'valency' (mood) of the emotion (-1 is

Figure 6.3: The SMAT 2DES Emotion Selector.

The 2DES Emotion Selector tab (Figure 6.3) allows the user to select a point on the 2DES and adjust the piece of music to that point. The adjustment is made assuming the original piece lies at the centre of the 2DES; the `(x,y)` value is then calculated and passed to `Adaptive.adjust()` appropriately.

6.3 Gun Man: Struggle Eternal

Gun Man: Struggle Eternal is a short platformer game created to demonstrate the use of ACRONYM in games. While ACRONYM is applicable to many genres of games, time constraints and the need for simplicity made me choose a 2D platformer as the demonstration game. *Gun Man* is written in Java using the GTGE game engine[16]. GTGE is fairly well-featured and is licensed under the LGPLv3, allowing me to inspect the source code where necessary.

In *Gun Man*, the player controls a hero character (the 'gun man' of the title) who can move, jump, and shoot. The aim of the game is to progress from left to right in the game world, jumping over

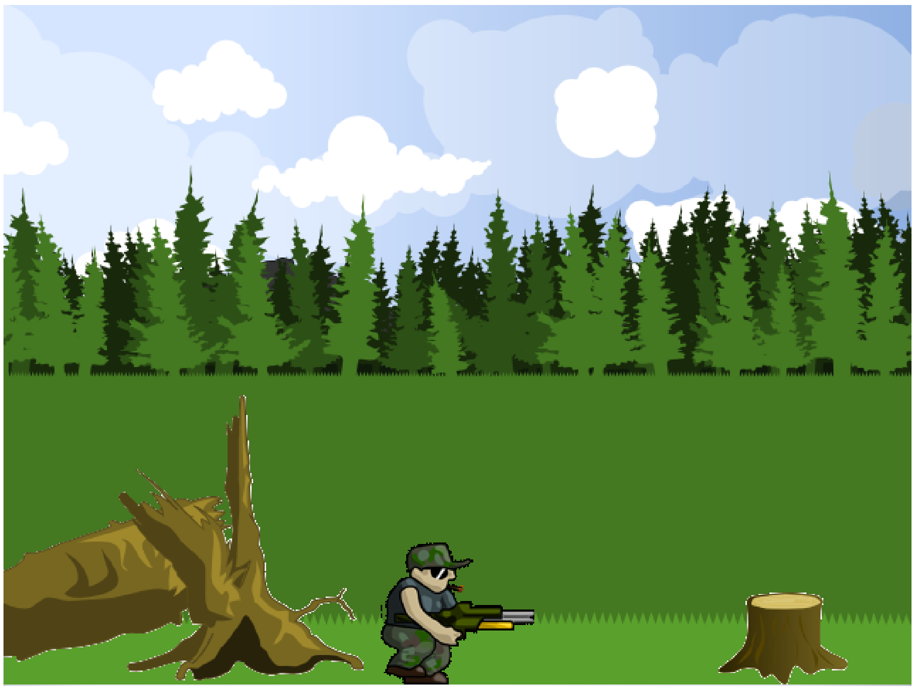


Figure 6.4: The ‘countryside’ area of the *Gun Man* game world.

various obstacles and killing enemy characters along the way. At the end of the level, the hero is rewarded with a trophy and wins the game.

6.3.1 Musical Transformations

The game world for *Gun Man* has three distinct areas: a ‘countryside’ area, with obstacles such as fallen trees; an ‘urban’ area, filled with looming skyscrapers and parked cars; and a ‘warehouse’ area, with boxes and barrels to navigate. The game detects when the player has crossed each area, using ACRONYM to alter the music over a few seconds. As the player progresses through the world, the music gets darker and more melancholy.

The game music also reacts to the presence of enemies. When an enemy appears on screen, the music is adjusted by (-0.2, 0.2) on the 2DES: a little more excited, a little less positive. This happens over the space of a second, as the presence of an enemy is a more sudden change requiring the player’s immediate attention. Once the enemy is killed or moves off the screen, the music returns to normal.

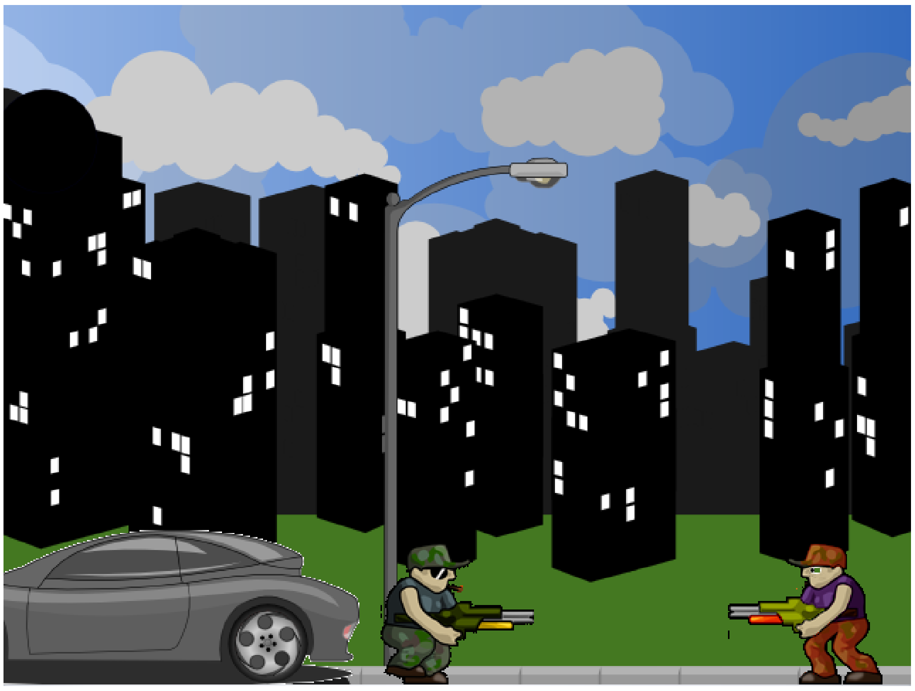


Figure 6.5: The ‘urban’ area of the *Gun Man* game world. The character to the right is an enemy.

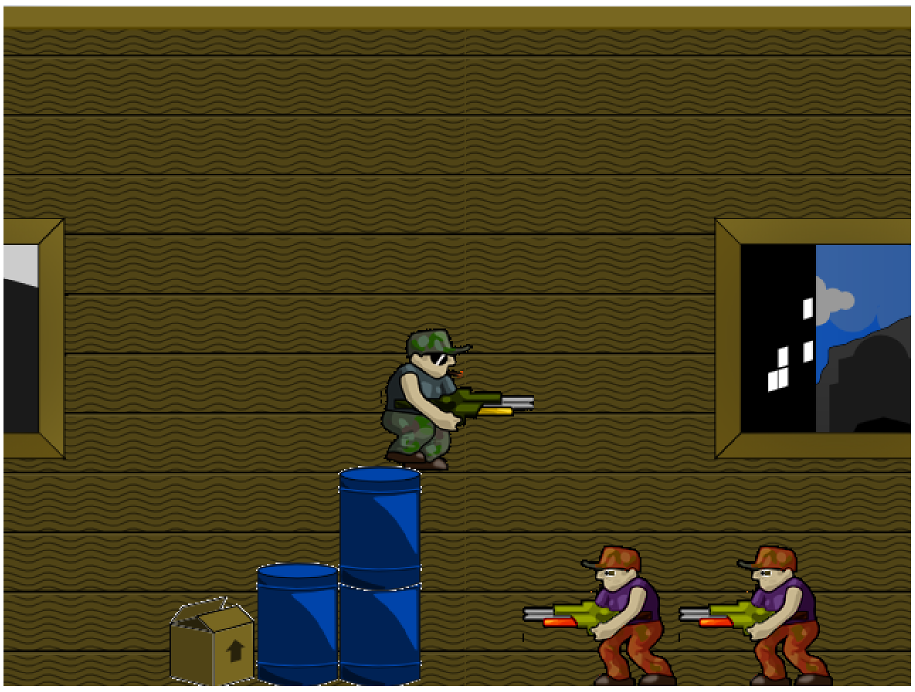


Figure 6.6: The ‘warehouse’ area of the *Gun Man* game world. Note the boxes and barrels, a staple of level design in games the world over.

Chapter 7

Testing and Experiments

This chapter details the various tests and experiments carried out to determine the abilities and efficiency of ACRONYM.

7.1 Emotion Recognition I

7.1.1 Method

An experiment was carried out to determine whether the transformations performed by ACRONYM did indeed influence the perceived emotion of the piece. Participants were directed to the web page reproduced in Appendix D.1 and asked to plot the perceived emotion points for an excerpt from Scott Joplin's *The Entertainer*, and for eight variations of the same excerpt, on a labelled copy of the 2DES. Each quadrant of the 2DES was represented by two variants, one with only a slight adjustment, and one with a more extreme adjustment. The variations were presented in a random order, but consistent between all participants. Details of the transformations for each variation can be found in Appendix D.2.1.

In the graphs and commentary that follows, the quadrants are referred to as (clockwise from top left): 1 - *Angry*, 2 - *Happy*, 3 - *Tender*, 4 - *Sad*.

7.1.2 The Original

In [29], Livingstone and Brown performed a similar experiment using two pieces: a Mozart string quartet, and a fragment of music from *The Legend of Zelda: Ocarina of Time*[39]. Their results, while encouraging, suffered somewhat from the fact that they asked their participants to rate the adjustment of emotion relative to the original pieces, which were said to be located at (0,0). This caused problems as both of the original pieces were fairly happy even before being altered; as a result, participants expressed difficulty in selecting quadrant 1 ('angry').

I decided to take advantage of this experience for my experiment and asked my participants to additionally plot a point on the 2DES representing the perceived emotion of the original, unaltered version of *The Entertainer*. Collating the results gave the combined plot shown in Figure 7.1.

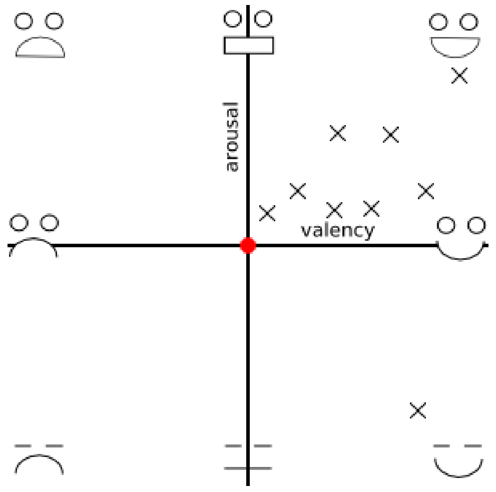
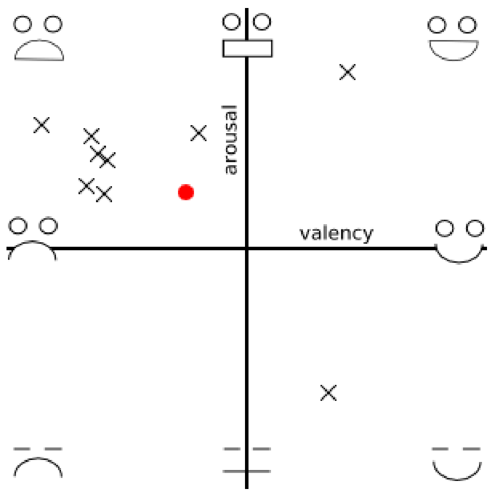


Figure 7.1: Combined plot of all perceived emotions for the original, unaltered version of *The Entertainer*. ACRONYM assumes the sample piece is located at dead centre, marked with a red circle on the graph.

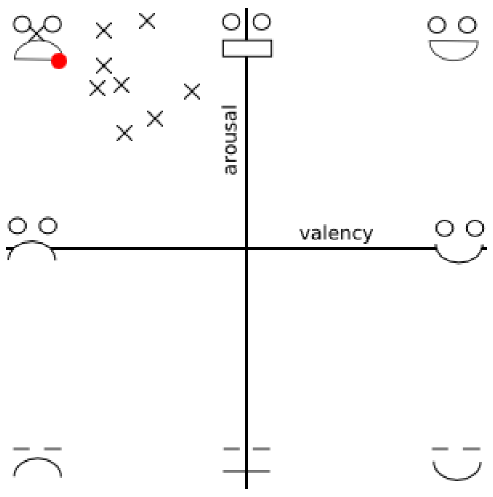
Unsurprisingly, given the piece, most participants placed a mark in the 'happy' quadrant. ACRONYM considers the original piece to be placed at (0,0) on the 2DES and performs its transformations relative to this point. As we shall see, this initial skew does have some effect on the results that follow.

7.1.3 Variations 1 and 5 - Angry

Figure 7.2 shows that these variations are reasonably effective: most of the results are concentrated in the first quadrant. There is a slight shift upwards and left in the 'slightly angry' version, and both variations are more centred within the quadrant than they should be. Relative to each other, Variation 5 is more aroused than Variation 1, but there was little perceived difference in valency. Both variations turned out to have quite a high tempo (87BPM and 127BPM, up from 72BM), so it is likely that this contributed to Variation 1's lower-than-required valency; it is also probable that the very high tempos distracted listeners from the other musical qualities, resulting in a more centralised valency figure for Variation 5.

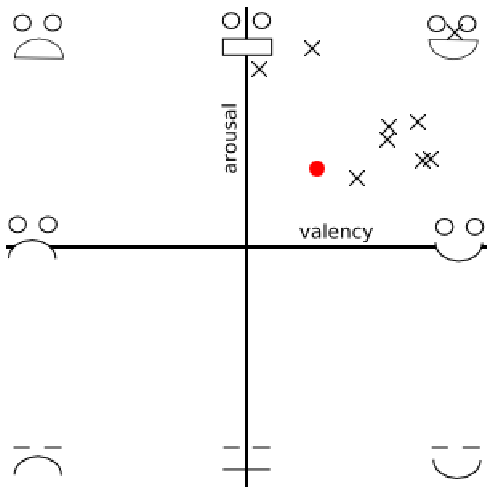


(a) Variation 1 – Slightly Angry

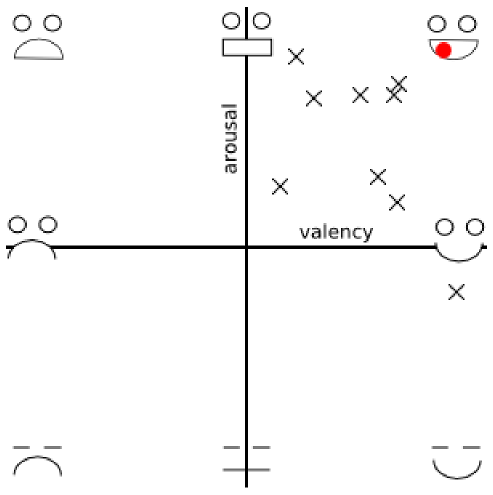


(b) Variation 5 – Very Angry

Figure 7.2: Combined plots of all perceived emotions for the ‘angry’ versions of *The Entertainer*. The point given to ACRONYM when creating the piece is indicated with a red circle on each graph.



(a) Variation 3 – Slightly Happy



(b) Variation 7 – Very Happy

Figure 7.3: Combined plots of all perceived emotions for the ‘happy’ versions of *The Entertainer*. The point given to ACRONYM when creating the piece is indicated with a red circle on each graph.

7.1.4 Variations 3 and 7 - Happy

Both happy variations (Figure 7.3) have approximately the same average point but Variation 7 has a higher standard deviation. Variation 7 suffered somewhat from overzealous note removal, resulting in a piece which had over half of its notes removed. This gave a somewhat stilted, haphazard effect which participants found difficult to place.

7.1.5 Variations 4 and 6 - Tender

The tender variations of the piece (Figure 7.4) have the largest spread of points and Variation 6 in particular deviates particularly far away from the intended point. The very slow tempo (45BPM) coupled with the removal of some notes gave the piece a bored, neutral feel, as opposed to a feeling of tenderness and contentment. Variation 4 was not as badly affected by this, although it did cross the valency line in a couple of instances due to the happy emotion of the original piece.

7.1.6 Variations 8 and 2 - Sad

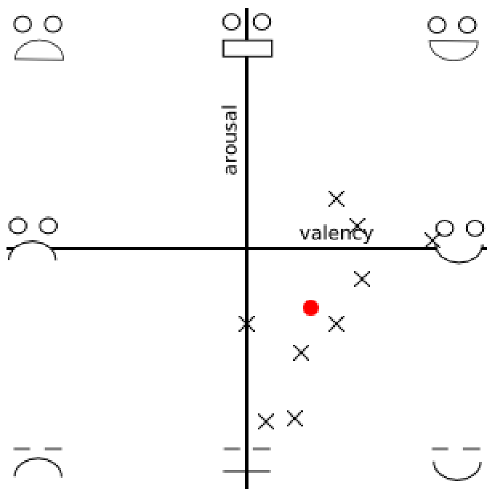
The variations of the piece set in the sad quadrant (Figure 7.5) have fairly reasonable participant results. Variation 8 was only slightly slower than the original (64BPM) and due to the original's increased arousal, a couple of participants marked it above the valency line. Variation 2, however, displayed the lowest standard deviation and the closest concentration around the intended 2DES point. This was due to the very morose slow tempo (42BPM) which provided a stark contrast to the original piece.

7.1.7 Analysis

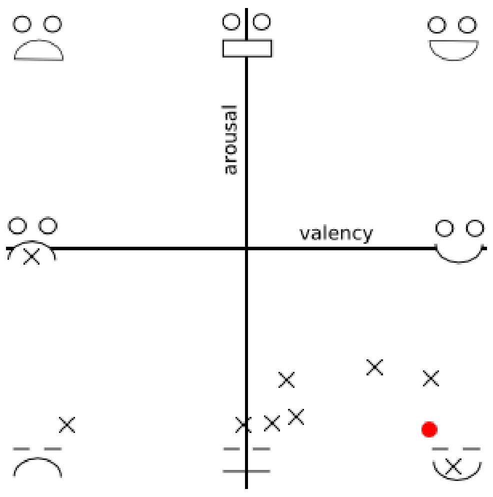
On the whole, ACRONYM's perceived emotion transformations were successful. There were, however, a number of factors which reduced the accuracy of the algorithm. These can be separated into two areas: problems with the source file, and problems with the transformation classes.

The Entertainer is quite a happy and lively piece, even without any transformations being applied. As shown in Figure 7.1, most participants placed the piece in the second quadrant, some even choosing to place it at fairly extreme arousal levels. This caused problems with variations supposed to be in the 'tender' quadrant, as the high initial arousal tended to pull the results up over the valency line.

The other, more subtle issue with *The Entertainer* as a source piece is that it is heavily syncopated. This does not work well with the *Harmoniser* (§5.2.8), which uses the first half of the beat for chord identification purposes. As a result, it is more than likely that the *Harmoniser* misidentified at least some chords, resulting in the addition of notes not in harmony with the beat's chord.

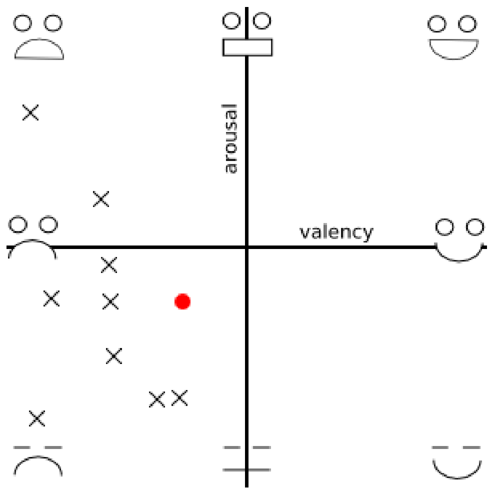


(a) Variation 4 – Slightly Tender

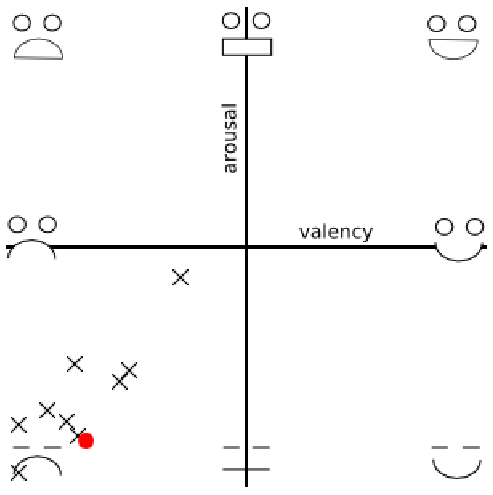


(b) Variation 6 – Very Tender

Figure 7.4: Combined plots of all perceived emotions for the ‘tender’ versions of *The Entertainer*. The point given to ACRONYM when creating the piece is indicated with a red circle on each graph.



(a) Variation 8 – Slightly Sad



(b) Variation 2 – Very Sad

Figure 7.5: Combined plots of all perceived emotions for the ‘sad’ versions of *The Entertainer*. The point given to ACRONYM when creating the piece is indicated with a red circle on each graph.

It is also likely that the settings of the transformation classes caused problems for the participants. The range of tempo scales covered by the 2DES is 0.5 to 2: in the slowest case, the piece will be at half speed, and in the fastest case, the piece will be at double speed. This very fast upper boundary seems to have caused problems in the 'angry' quadrant, with a highest tempo of around 130BPM. This is not a sensible tempo and the results indicate that the tempo range is too wide. Similarly, Variation 7 ('extremely happy') suffered from large amounts of note removal, to the point where participants did not have enough notes to be able to accurately rate the emotion of the piece.

The results of the first experiment indicate that ACRONYM is generally capable of altering a piece towards a perceived emotion, but there is still certainly room for improvement. We will examine such improvements in the next section.

7.2 Emotion Recognition II

After the first set of experiments I set out to improve ACRONYM's transformation classes in an attempt to get more accurate results. I chose a two-pronged *modus operandi*: firstly, to adjust the scale of the transformations; and secondly, to choose a better source piece.

I began by adjusting the scale of the transformations, working from the results in Section 7.1.7. The range of valid tempos was narrowed to [0.75, 1.5], and I found a bug in the adjustment for the *Harmoniser* which set the lower bound for removing notes to 0 rather than 0.5 as initially required. I fixed this, and then adjusted the *Harmoniser* scaling parameters to [0.75, 1.5].

I then searched for a new piece of music to use. Having learnt from my first set of experiments, I tried to find a piece with a medium tempo, but still fairly sedate and unassuming in style. After some searching, I settled on Franz Schubert's *Moments Musicaux, No. 3, F Minor (Op. 94)*, which meets the criteria: 90BPM, and a repetitive, rhythmically unexciting melody line. To confirm my judgement, I ran a short experiment asking participants to rate the piece on the 2DES. Satisfied with the results (Figure 7.6), I began a similar experiment to my first, using similar points on the 2DES. The resulting transformations are detailed in Appendix D.2.2.

Moments Musicaux No. 3 is in a minor key, in contrast to the first experiment. It is interesting to note that despite this, many participants still rated the piece as happy. Overall, while there is some variance within the results, they are roughly centred, which provides a good control for the other variants to be compared against.

7.2.1 Variations 4 and 6 - Angry

There is a definite difference in perceived emotion between the two 'angry' variants (Figure 7.7): most participants identified one variation as 'more angry' than the other. The reduction in tempo variance seems to have helped participants rate valency more objectively without being distracted by very high tempos.

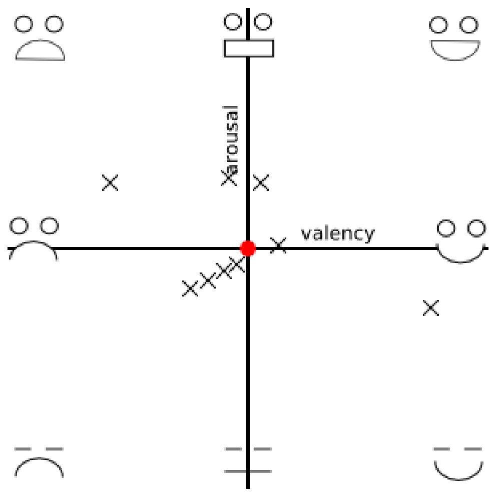
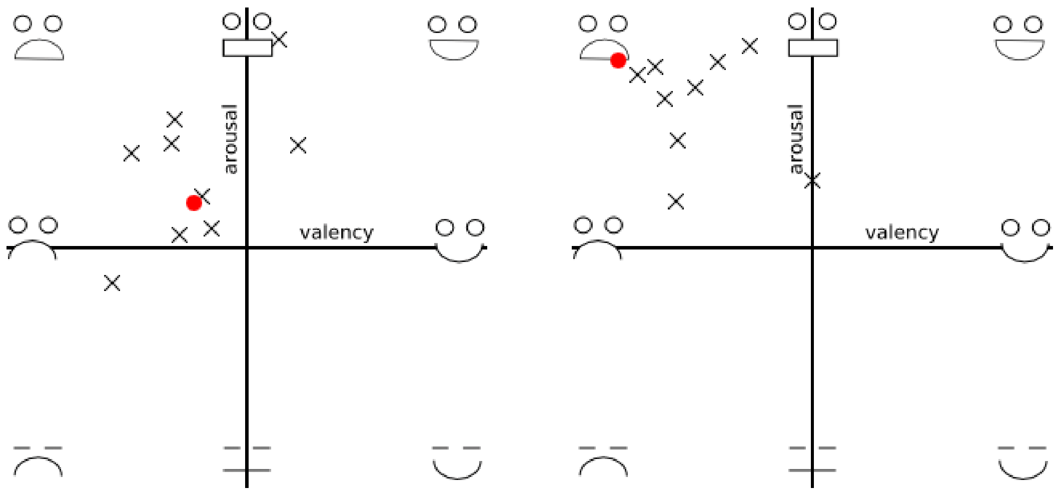


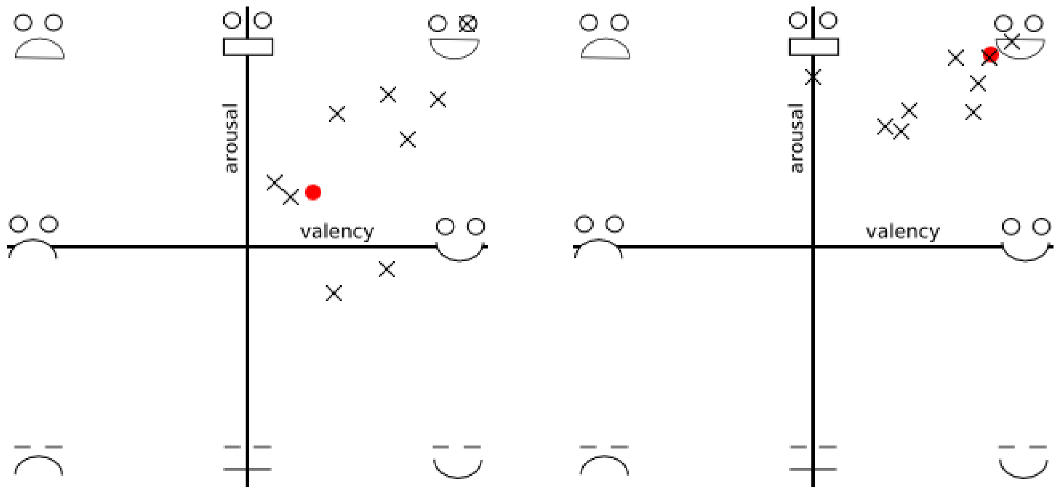
Figure 7.6: Participant results for an excerpt from an unaltered rendition of *Moments Musicaux No. 3.*



(a) Variation 4 – Slightly Angry (b) Variation 6 – Very Angry

Figure 7.7: Combined plots of all perceived emotions for the 'angry' versions of *Moments Musicaux No. 3.* The point given to ACRONYM when creating the piece is indicated with a red circle on each graph.

7.2.2 Variations 5 and 2 - Happy



(a) Variation 5 – Slightly Happy (b) Variation 2 – Very Happy

Figure 7.8: Combined plots of all perceived emotions for the 'happy' versions of *Moments Musicaux No. 3*. The point given to ACRONYM when creating the piece is indicated with a red circle on each graph.

Results for the 'happy' variants (Figure 7.8) are somewhat improved on those from the first test. The level of note removal had been dropped, effectively giving participants more notes to judge the pieces by. There is still however a fairly large standard deviation, especially with the mild variant.

7.2.3 Variations 1 and 7 - Tender

Perhaps due to the fairly upbeat nature of the original piece, the 'slightly tender' variant (Figure 7.9a) was rated far more in the 'happy' quadrant than the 'tender' quadrant. This is unsurprising, as the difference between the transformations for Variations 5 and 1 are very small (see Appendix D.2.2). The 'very tender' variant fared somewhat better, although points tended to be further up and left of the desired point.

7.2.4 Variations 8 and 3 - Sad

The 'sad' variants (Figure 7.10) performed better than in the first experiment, with a more marked distinction between the mild and extreme versions and a far smaller standard deviation.

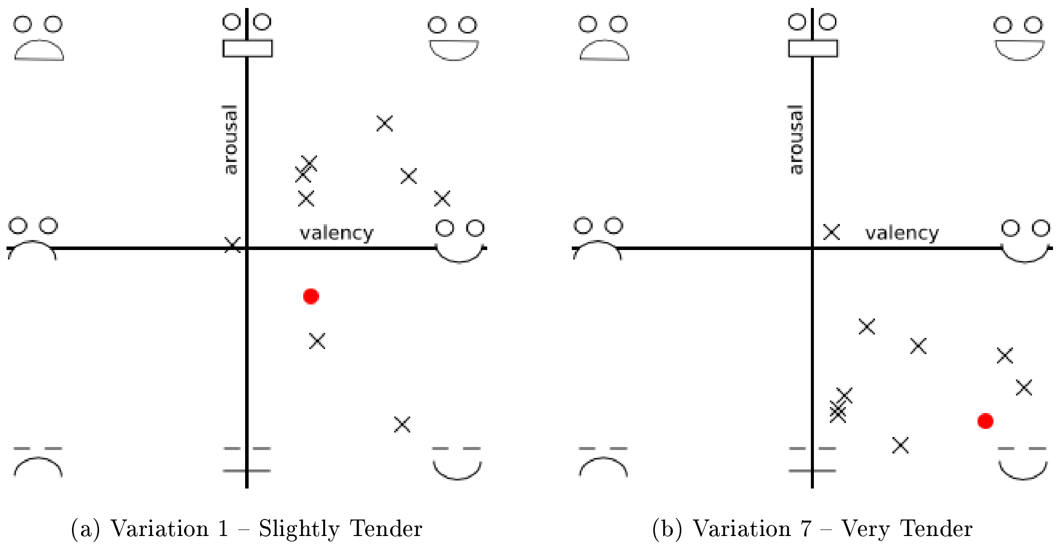


Figure 7.9: Combined plots of all perceived emotions for the ‘tender’ versions of *Moments Musicaux No. 3*. The point given to ACRONYM when creating the piece is indicated with a red circle on each graph.

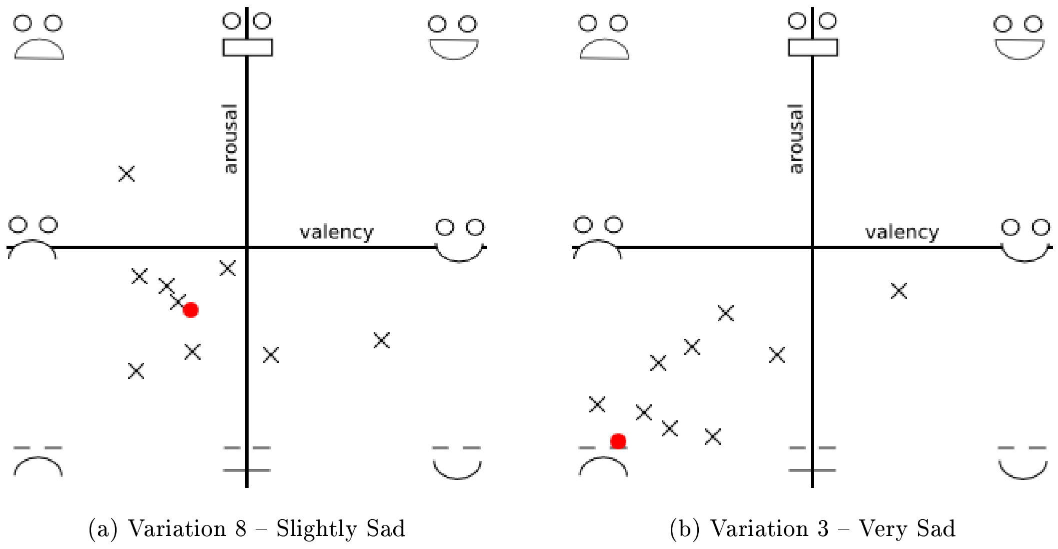


Figure 7.10: Combined plots of all perceived emotions for the ‘sad’ versions of *Moments Musicaux No. 3*. The point given to ACRONYM when creating the piece is indicated with a red circle on each graph.

7.2.5 Analysis

On the whole, the second experiment fared better than the first, thanks to the reduction in the 'scale' of the transformations. However, the extreme transformations were often perceived as being towards the centre of the quadrants, indicating that in this case the scale was perhaps *too* narrow. The small differences in scaling particularly affected Variation 1. This indicates that scaling parameters should be set on a per-piece basis, and that a single scaling range for all pieces is insufficient to reliably transform music to a given emotion.

Additionally, some indication of the initial emotional value of each piece would be useful, instead of assuming that all pieces are neutral to begin with and transforming them relative to the original; this would improve ACRONYM's performance in transforming pieces to absolute (rather than relative) points on the 2DES.

7.3 Performance Testing

I noticed during the development of *Gun Man* that while the adaptation was taking place, a noticeable drop in framerate occurred. This was enough to overload a moderately powerful computer[1] and cause jitter in the graphics of the game. One of the requirements were ACRONYM to be used in a commercial game would be a low overhead: specifically, that the framerate of the game should not drop lower than 30 frames per second (fps). ACRONYM's adaptation feature was frequently bringing *Gun Man* to around 10-15fps for the duration of the adaptation (five to ten seconds for an average piece), so I decided to investigate further.

I installed the Eclipse Test & Performance Tools Platform on my local machine and ran the included profiler. This showed that the vast majority of CPU time was being taken up by the binary search algorithm, used to locate each event in the currently-playing sequence from the copy (see §5.2.3). Further inspection of the profiler's data revealed that the call `assert(isSorted(midisequence))` was the culprit - each call to the binary search algorithm was first ensuring that the sequence is sorted, which required examining every event in the sequence. This made a single run of the *Adapter* have $\Omega(n^2)$ efficiency for a sequence of n events. The call itself was a remnant of a debugging session before I knew the internal format of a Java MIDI sequence was *guaranteed* to be sorted. A second profiling test with asserts turned off cut the time spent adapting the sequence to about a fifth.

Despite this, the framerate drop still occurred, just for not as long. Adding a `Thread.yield()` call after the processing of every event in the *Adapter* allowed Java to schedule CPU time for the game more regularly, improving the framerate to an acceptable level.

[1]AMD Athlon XP3000+, 1GB RAM, running Windows XP SP2

Summary

Testing shows that ACRONYM's emotion selection mechanism works well, with the majority of participants picking the correct quadrant. ACRONYM is reasonably good at creating perceptible differences between the same emotion, although intelligent scaling based on the qualities of the input piece would help greatly in this regard. With a good scheduler and by giving the game loop thread priority, ACRONYM's dynamic music adaptation has no noticeable impact on game fluidity and does not cause the game to drop below 30fps.

Chapter 8

Conclusions and Further Work

8.1 Conclusions

The objectives of this project were to:

1. Adapt an entire MIDI file to shift its perceived emotion in a given direction by a set amount:
    (a) Gain knowledge of various music-emotion rules and encode them into the system;
    (b) Cater for a wide range of emotions, and to produce pieces with the same perceived emotion as the intended emotion;
    (c) Adjust the degree of the perceived emotion, e.g. be able to create at least two distinct 'angry' tracks, where one sounds angrier than the other, but where both sound angrier than the input track.
2. Play back a MIDI file and 'push around' its perceived emotion in real-time:
    (a) Be able to cope with applying the rules for each emotion in real-time, *while leaving enough CPU time for the game to run on*;
    (b) Be able to handle the transition between emotions gracefully and unobtrusively.

The two experiments carried out (Sections 7.1 and 7.2) show that ACRONYM is more than capable of creating versions of pieces with varying perceived emotions. It is generally good at creating pieces that are recognised as being in the correct quadrant of the 2DES, and makes a reasonable attempt at creating a specific emotion. However, this is a very subjective issue and no system will be 100% correct in this regard, simply because of the differences in the way different people perceive emotion. ACRONYM has shown itself to be capable of creating music which spans most of the range of the 2DES, which covers most (if not all) of the basic emotions. The call to `adjust()` can also be given a time parameter to create a gradual change towards the new emotion over a number of seconds.

Finally, the system is also able to adapt a MIDI file fast enough to be able to run in real-time, and is threaded in such a way that it does not hog system resources, allowing another application (for example, a 2D platform game) to run smoothly (Section 7.3).

8.2 Further Work

ACRONYM is far from feature complete. A (partial!) list of features which could be added is as follows:

Analysis Pass Currently, ACRONYM does little to no analysis of the piece before playing, working instead directly with the MIDI events at a low level. An analysis pass through the file on load would provide many opportunities:

- **Phrase Detection** ACRONYM cannot currently tell the structure of a piece of music and how groups of notes are related. Better knowledge of where phrases are would allow for more complex articulation transformations, such as applying a crescendo and diminuendo over the space of a single phrase.
- **Melody Detection** Being able to extract the melody line – and indeed harmony lines – would greatly improve the treatment of minor sevenths, increasing the frequency of picking the correct (sharpened or flattened) version.
- **Key Detection** ACRONYM performs particularly badly when given a MIDI file with incorrect or completely absent Key Signature events. It is heavily reliant on the presence of these events to determine the key of the piece. An analysis pass could note the frequency of accidentals in the piece along with its possible chord progressions and use this information to detect the key of the piece.
- **Chord Detection** ACRONYM's current technique of examining only the notes in the first half of a beat for the purposes of detecting chords works well for classical music but tends to fail when applied to syncopated music such as *The Entertainer*. Detection of chords and paying close attention to their placement within a bar could help identify the dominant chord of the beat and bar, even if the notes were to appear in odd places.

Instrumentation Changes Re-orchestrating the piece to use different instruments would certainly help change the perceived emotion, as different instruments tend to have different emotional connotations.

Part Creation Adding texture by creating entirely new parts would be a daunting task, but moving from a light solo to a full orchestral swell would provide a broad range of possible emotions, and also act as an intensifier for any given emotion.

Intelligent Harmonisation Better key and melody detection would pave the way for more intelligent harmonisation. Notes could be added to ensure they do not cause disharmony with others and fit the overall flow of the music; note removal could be done with grace, preserving the theme of the piece and extending the lengths of other notes to fill in the gaps.

Digital Signal Processing The games industry moved away from the MIDI file format for background music in favour of waveform representations such as MP3 and OGG. MIDI uses any available synthesiser on the system, which can lead to poor sound quality on older systems. Using DSP to take a waveform file as input would be incredibly difficult, but if possible (or if all parts were separated into their individual instruments and analysed separately), much higher sound quality would be available.

Finally, the most important change that could be made to ACRONYM would be to port it to C++, the language of the games industry. This could potentially spur development and allow it to be used in commercial games.

8.3 Concluding Remarks

It is hoped that this project demonstrates that an adaptive approach to background music in games is possible, and with further work, can provide a musically enlightening and emotionally sensitive addition to video games. With graphics nearing photorealism and physics simulation becoming more accurate daily, it is hoped that sound and music design will be the next area of innovation for the games industry, and that the use of music in games will mature and become more sophisticated in the near future.

Appendix A

A Brief Introduction To Music Theory

This appendix aims to give readers an introduction to basic music theory, familiarity of which is required to understand the principles behind this project.

A.1 Notation

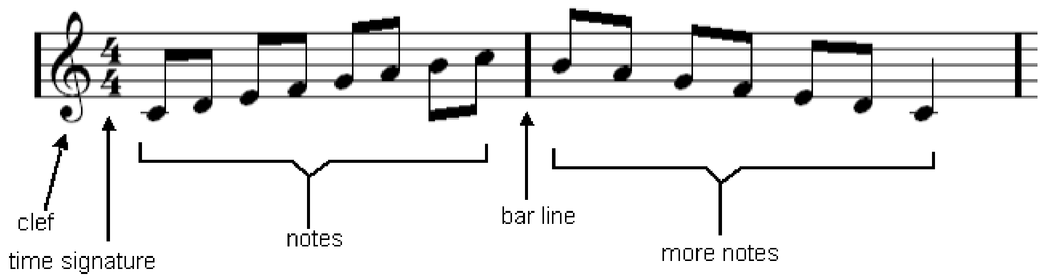


Figure A.1: The scale of C Major.

Figure A.1 shows a simple piece of music, represented using staff notation. It is made up of the following elements:

notes The markings to indicate which pitch is to be played. Notes consist of a head (the circular part), which may be filled or hollow, and usually (but not for certain lengths of notes) a stem (the line extending from the head). Crudely speaking, notes without stems are longer than those with stems, and are always hollow. Hollow notes are longer than filled notes, and the more additional lines a note has attached to its stem, the shorter the note is. Further knowledge of note names, lengths, and notation is not required for this project, and as such is omitted.

staff The five horizontal lines on which the notes are written. Notes with their heads positioned lower on the staff are lower pitched than those placed higher on the staff.

clef Written at the beginning of every line of music notation. Tells the performed at what pitch the following notation is to be played.

bar line The staff is divided into *bars* using bar lines, allowing the musician to refer to bar numbers to indicate different sections of the piece.

time signature The numerator of the time signature tells the performer how many *beats* are in a bar, and the denominator defines how long each beat is. The denominator is always a power of two; higher numbers indicate shorter notes (the number is in terms of divisions of a note called a *semibreve*). The piece of music in Figure A.1 has a time signature of $\frac{4}{4}$, which indicates there are four (the numerator) crotchet (the denominator) beats in every bar. The time signature is written once at the beginning of the piece, and only written again if it changes.

There are also other optional notational elements:

accidentals ($\flat$, $\sharp$, $\natural$) A flat ($\flat$) lowers the note by a semitone (one step) and a sharp ($\sharp$) raises the note by one semitone. Accidentals stay in effect for all notes of the given pitch until the end of a bar, so a note may be returned to its 'normal' state by using a natural ($\natural$), which reverts the change.

key signature Informs the musician that the given notes are to be assumed to be sharpened or flattened unless otherwise indicated (with a natural, see Figures A.2 and A.3). In Western music, key signatures are standardised in order to also indicate the *key* of the piece (see the next section).

A.2 Scales and Keys

The notes in Figure A.1 are arranged in a special order: they make a *scale*. Most scales in Western music consist of eight notes, with special names. The notes, in ascending order, are referred to as: *tonic, supertonic, mediant, subdominant, dominant, submediant, leading note* (the final note is the initial note plus one octave, and is thus the *tonic* once again). There are various different types of scale, each corresponding to a certain set of *intervals* (distances) between the notes of the scale:

major The set of distances `{2, 2, 1, 2, 2, 2, 1}` (and the reverse when descending). The the scale in Figure A.1 is a major scale starting on the note C: it is the *C Major scale*. In

Western music, this scale (starting on any note) is associated with happiness and positive mood.

minor In Western music, the minor scale is associated with sad, angry, or fearful music, i.e. negative mood. It is characterised by the flattening of the 3rd, 6th and 7th notes in the scale. However, there are various combinations of these flattenings, leading to different versions of the minor scale:

harmonic The set of distances `{2, 1, 2, 2, 1, 3, 1}`. The interval between the sixth and seventh notes of this scale is an *augmented second* (three steps), which can be seen as awkward and especially difficult when singing. This led to the widespread use of the *melodic minor* scale.

melodic The melodic minor scale is different ascending and descending. Ascending, the set of distances is `{2, 1, 2, 2, 2, 2, 1}` (flattened seventh only); descending, it is `{2, 2, 1, 2, 2, 1, 2}` (from the top; flattened sixth only). Figures A.2 and A.3 show the C melodic minor scale, the latter including its key signature.

natural The natural minor scale is the set of distances `{2, 1, 2, 2, 1, 2, 2}`, which corresponds to flattening both the sixth and seventh. This therefore is essentially the descending version of the melodic minor scale.

others There are also other scales with different interval patterns, such as the *mixolydian* and *aeolian* scales. These are far less common in Western music, despite falling into the same category of *diatonic* scales (which consist of five two-step intervals and two maximally separated one-step intervals) as the major and minor scales.

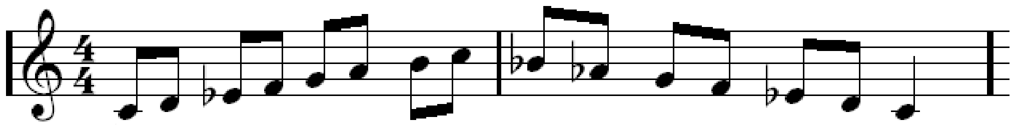


Figure A.2: The scale of C melodic minor.

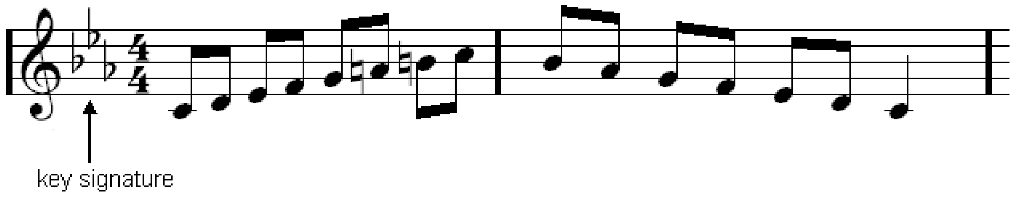


Figure A.3: The scale of C melodic minor, using its key signature.

Pieces of music are often said to be in a certain *key*. This means they use the notes of a particular scale, so a piece in the key of C Major uses primarily the notes in that scale. The key signature of a piece can help to identify the key: for instance, a piece using a major scale with a key signature of one flat (which is always B♭ due to the standardisation of key signatures) can be deduced to be in the key of F Major, as the F Major scale has B♭ as its only accidental. However, the same piece in a *minor* scale (still with a key signature of one flat) is in the key of D Minor, not F Minor.

This may seem odd, but is easily explained on further inspection. The F Minor scale is the F Major scale (which has one flat), with the third, sixth, and seventh notes all flattened – making four flats in all. As it happens, it is D Minor which shares the same key signature as F Major, and these keys are the *relative major/minor* of each other. The relationship between F Major and F Minor is described as being the *parallel major/minor*.

A.3 Performance Markings

Staff notation also traditionally employs various performance markings, mostly derived from Italian terms, to indicate that the musician should play the notes loudly, quietly, fast, slow, or shorter or longer than their usual length. An explanation of the various signs and symbols involved is not necessary for the purposes of this project; a glossary of the terms used in this report is provided on the next page.

Glossary of Musical Terms

articulation The direction or performance techniques that affect the transitions between notes. This includes directions such as staccato and legato.

BPM the number of beats (the type of beat being determined by the denominator of the time signature) per minute - an indication of how fast the piece should be played.

chord Two or more notes played at the same time.

chord progression A sequence of two or more chords played in turn.

dynamics The volume markings associated with a piece, e.g. *crescendo* (gradually getting louder) and *diminuendo* (gradually getting quieter).

harmony The use of different pitches simultaneously: the 'vertical' aspect of a piece of music.

legato *from Italian, lit. tied together.* Notes are to be played smootly.

ritardando *from Italian ritardare, to slow down.* A gradual decrease in tempo.

rubato *from Italian, lit. stolen time.* Slight tempo fluctuations in a piece, performed at the discretion of the soloist or conductor.

staccato *lit.* detached. Notes are played abruptly and are shortened, with silence filling the extra space between notes.

syncopated Syncopated music has the musical stresses off, rather than on the beat. It is frequently used in jazz music.

tempo *from Latin tempus, meaning time.* The speed of a piece of music.

timbre *from French.* The quality of a note that distinguishes it from the same note played by another instrument.

triad A chord consisting of three notes, usually spaced as tonic, mediant, and dominant.

Appendix B

The MIDI Specification

This chapter aims to give the reader an understanding of the MIDI specification, knowledge of which is crucial to understanding the underlying technology ACRONYM is built on. We begin with a brief look at what MIDI is, its uses, and then examine in more detail the structure of a MIDI file and the way music, and its properties, is encoded.

B.1 Introduction

MIDI (Musical Instrument Digital Interface) is a protocol which enables musical instruments, equipment, and computers to synchronise and communicate with each other. Simply put, MIDI specifies a set of messages which MIDI-compatible pieces of equipment use to communicate. These messages are then sent over MIDI cables in real time, or stored in a Standard Midi File (SMF) for later playback. MIDI encodes the act of playing an instrument at an abstract level, rather than attempting to duplicate the waveform produced; this results in far smaller file sizes for an SMF than a file which represents the waveform.

MIDI is both a hardware and a file specification: the hardware specification is concerned with the connectors used to connect MIDI-compatible equipment together, and the file specification details the format of an SMF. MIDI-compatible equipment comes with MIDI In, MIDI Out, and MIDI Thru ports, for input, output, and a special second 'daisy-chaining' output (used to pass on an instrument's MIDI In messages to another instrument) respectively.

The action of playing a single note on a MIDI keyboard can be described as follows:

1. The musician presses the key. The keyboard sends a `Note-On` message, containing the pitch and volume of the note (both encoded as numbers from 0 to 127), on its MIDI Out port.
2. The musician releases the key. The keyboard sends a `Note-Off` message, containing the pitch and (optionally) decay time of the note (both encoded as numbers from 0 to 127), on its MIDI Out port.

This is, essentially, the basis of MIDI: a stream of `Note-On` and `Note-Off` messages, known as 'events' in MIDI terminology. These events are received and processed on one of 16 MIDI Channels, each of which can be set to sound as a different instrument if the synthesizer supports it. Any instrument receiving a `Note-On` event on its MIDI In port for a particular channel sounds a note at the given pitch and volume, using an instrument sound from its pre-programmed sound bank corresponding to the channel the message was received on. It continues sounding the note until it receives a `Note-Off` event for the same pitch on the same channel.

B.1.1 General MIDI

General MIDI (GM) is an additional specification that MIDI synthesizers may support. It defines (amongst other things):

- A pre-set list of 127 instrument sounds (known as 'patches') the synthesizer must be able to produce. The MIDI event `Program Change` is then used to change the instrument the synthesizer must use for a given channel.
- A minimum number of notes that can be played at the same time – at least 16 melodic and 8 percussive notes.
- The use of MIDI Channel 10 as a percussion channel, with specific percussion instruments assigned to each note played on that channel.
- Extra controllers for pitch modulation, panning, and sustaining notes. (A pitch bend controller was defined in the original MIDI standard.)

Most modern MIDI equipment supports the GM standard, and as such this report will use the word 'MIDI' to refer to the MIDI standard *including* the extensions defined by GM.

B.2 MIDI Messages

MIDI messages can be grouped into two categories: *events* and *meta-events*. *Events* are the messages sent across MIDI cables, allowing MIDI equipment to synchronise and control each other; *meta-events* are special event types saved in SMFs only, usually containing notative or performative instructions for MIDI software to display. Meta-events are generally optional; predefined default values are used in their absence as stated below. The lists below are not exhaustive, and focus mainly on the messages which are of use in this project.

B.2.1 Events

All events consist of a *status* byte to identify the event type and channel, and zero or more data bytes. Channel values range from 0-15 (0-F in hex); all other values range from 0-127 unless otherwise stated.

`Note-On` **0x9*ch* *note velocity*** Play note number *note* on channel *ch* at volume *velocity*. A `Note-On` message with a *velocity* of zero is treated as a `Note-Off` message for the given pitch. If two `Note-On` messages are received for the same note and channel without a `Note-Off` between them, it is up to the instrument to decide whether to layer another voice over the top of the first note, or to cut off the first note and then retrigger it. Instruments that do not support variable velocities will transmit this message with a default velocity of 64.

`Note-Off` **0x8*ch* *note velocity*** Release note number *note* on channel *ch* at release speed *velocity*. Instruments are at liberty to use *velocity* information as they please. Instruments that do not support variable velocities will transmit this message with a default velocity of 64.

`Aftertouch` **0xA*ch* *note pressure*** Apply a vibrato effect to note number *note*. The amount of vibrato applied depends on the amount of *pressure* with which the musician is holding down the key.

`Controller` **0xB*ch* *controller value*** Set the controller numbered *controller* on channel *ch* to *value*. Controllers control audible properties such as modulation (controller 1), panning (controller 10), and sustenuto (controller 66). The special controller `All Notes Off` (controller 123) can be used to release any notes for which `Note-On` messages have been received; the value byte is not used in this case.

`Program Change` **0xC*ch* *instrument*** Change the instrument sound on channel *ch* to *instrument*. The *instrument* is one of the 127 GM patches.

`Pitch Wheel` **0xE*ch* *byte1 byte2*** Adjusts the channel's pitch by the value given by *byte2byte1* (i.e. the value is passed in a little-endian format). A value of 0x2000 indicates the pitch wheel is centred and therefore no adjustment is to be made. The GM specification recommends that the entire range of possible `Pitch Wheel` messages covers ± 2 semitones.

`MIDI Start` **0xFA** Starts playback of a song or sequence from the beginning.

`MIDI Stop` **0xFC** Stops playback of a song or sequence. The device must remember the position within the song in case of a MIDI Continue message arriving later.

`MIDI Continue` **0xFB** Continues playback from the current song or sequence position.

B.2.2 Meta-Events

Meta-events are special, non-MIDI events that are never sent to actual MIDI instruments. Instead they are used by MIDI software for notative or performative purposes. All meta-events begin with the status byte 0xFF.

`Copyright` **0xFF 02 *len text*** A copyright message of length *len* characters.

`Tempo` **0xFF 51 03 *byte1 byte2 byte3*** A tempo change, expressed in microseconds per quarter note. *byte1byte2byte3* expresses how long each quarter note should be, in microseconds. In the absence of any tempo events in a file, the tempo is assumed to be 120BPM.

`Time Signature` **0xFF 58 04 *num den metro 32nds*** A change of time signature. *num* and *den* are the numerator and denominator; the denominator is expressed as a negative power of 2, so *num=4 den=2* indicates a time signature of $\frac{4}{4}$. *metro* is the number of MIDI Clocks per metronome tick. There are 24 MIDI Clocks in every quarter note. *32nds* is the number of 32nd notes in a quarter note. In the absence of any time signature events in a file, the time signature is assumed to be $\frac{4}{4}$.

`Key Signature` **0xFF 59 02 *sf mi*** A change of key signature. *sf* is set to the number of sharps in the key; a negative number indicates flats. *mi* is 0 if the key is major, and 1 if the key is minor.

`End Of Track` **0xFF 2F 00** The end of the track. This event is *not* optional.

B.3 Standard Midi File Format

All SMFs begin with a header, which consists of an ID of `MThd`, a *length* byte (which is always 6), and then six further bytes. The first pair of these indicates which format the MIDI file is in (see below). The second pair of bytes indicates the number of tracks in this file. The final pair of bytes indicate the number of MIDI clock *pulses per quarter note* (PPQN). All byte pairs are stored in big-endian format.

Following the header is one or more `MTrk` chunks – the actual music tracks in the MIDI file, the number of which was stated in the header. An `MTrk` chunk begins with the ID of `MTrk`; the next four bytes indicate the length of the track (in bytes). The last element of an `MTrk` chunk is the data of the track: the actual MIDI messages, arranged in time order. Each message is preceded by a delta-time, indicating the number of clocks between the previous message and this one.

B.3.1 MIDI File Formats

There are three formats for SMFs, detailing the format and intended nature of the MTrk chunks:

Format 0 The MIDI file contains only one `MTrk`, containing MIDI data for all 16 channels.

Format 1 The MIDI file contains multiple simultaneous `MTrk`s, which are all started at the same time. The first `MTrk` in a MIDI file may contain only meta-events, in which case a tempo and time signature 'map' can be created from this track. The other tracks are usually separated into channels, although this may not always be the case.

Format 2 The MIDI file contains one or more sequentially independent tracks; the sequencer will cue up each track separately, potentially at different times. This format is relatively uncommon.

ACRONYM uses Format 0 MIDI files, as events for all channels need to be read and adjusted in order to ensure the system 'keeps up' with the MIDI file as it is being played. Source MIDI files are automatically converted to Format 0 on initial reading if necessary.

Appendix C

jMusic

This chapter outlines the Java library jMusic [51], explaining its purpose, structure, and use in this project. We will also examine the modifications made to the library that were made to suit ACRONYM's needs.

C.1 Introduction

jMusic is a Java programming library aimed at musicians, created by researchers at the Queensland University of Technology, Australia. It is released under the GPL and is designed to assist with computer composition. It includes methods for reading and writing to MIDI files via its own internal data structure, which is similar to the layout of a MIDI file, as well as a huge library of constants, giving meaningful names to MIDI pitches (for instance C♯ above Middle C can be referred to by the constants `CS4` and `DF4`), note lengths (`CROTCHET` and `QUARTER_NOTE` are meaningful to the jMusic internal format), and scales (for instance `MAJOR_SCALE`, `MELODIC_MINOR_SCALE`, `MIXOLYDIAN_SCALE` are all defined in terms of intervals starting from 0).

C.1.1 jMusic Data Structure

The jMusic internal data structure is divided into four parts: `Score`, `Part`, `Phrase`, and `Note`.

Score The top-level structure, containing a Vector of `Parts`. It also contains a title, a key signature, a time signature, and a tempo. Roughly corresponds to an entire MIDI file.

Part Contains a vector of `Phrases`, a title, a channel, and an instrument. Roughly corresponds to a single `MTrk` in a Format 1 MIDI file.

Phrase Each phrase contains a vector of `Notes`. Phrases are monophonic: they do not have any overlapping notes (chords must be separated into phrases, or else use the special class `CPhrase`). `Phrases` may start at a certain number of beats into the `Part`. There is no similar structure in a MIDI file.

Note Each `Note` contains information storing the note's pitch, volume, length, etc.

C.2 Modifications

During the development of ACRONYM, various modifications were made to the jMusic library to add missing functionality. These modifications required alterations to the jMusic functions `Read.midi()` and `Write.midi()`, as well as the creation of new data structures specific to the modifications. As a result, jMusic now preserves the following events in MIDI files:

C.2.1 Key Signatures

jMusic's handling of Key Signature meta-events was naïve at best. Any Key Signature event read in would overwrite the single key signature field in the `Score` class, meaning that the key signature given to the entire piece would be the last key signature in the file. Contact with the maintainer of jMusic indicated that this was indeed an incomplete feature, so as a result I extended the `Part` class to contain a list of key signatures for that part. The *Transposer* (§5.2.4) uses this list to determine the key of the current note, and modify it appropriately.

C.2.2 Time Signatures and Tempos

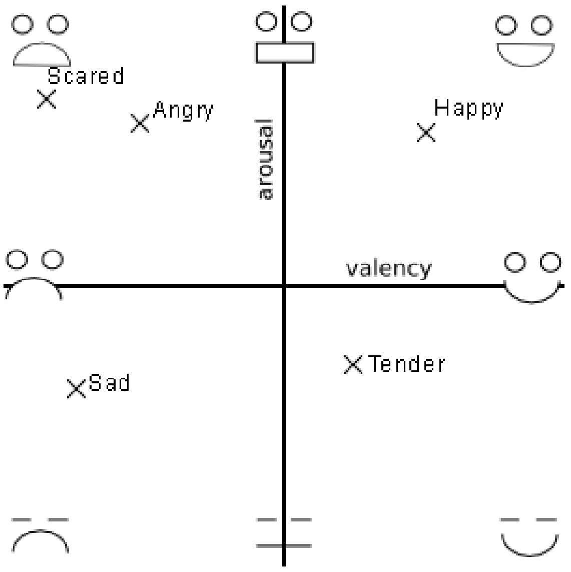
As with key signatures, jMusic overwrote a single field in *Score* holding the score's time signature. I subclassed `Part` and created a new class `MetaTrackPart`. Each `Score` has a single `MetaTrackPart` to store `Time Signature` and `Tempo` meta-events, much as in a Format 1 MIDI file. Any `Time Signature` or `Tempo` meta-events read when `Read.midi()` is called are placed into structures in the `MetaTrackPart`, containing details of the event and its position in the piece.

Appendix D

Experiments

D.1 Experiment Sheet

This experiment asks you to listen to nine different variations of the same piece of music, and mark on a graph the point which best describes the emotion of each variant.



Here is the graph. It plots arousal (excitedness) on the vertical axis against valency (the pleasantness of the emotion) on the horizontal axis. So positive emotions such as happiness and tenderness are on the right, and negative ones such as sadness or anger are on the left; similarly, 'extroverted' emotions such as happiness or anger are at the top, and 'introverted' emotions such as sadness and tenderness are at the bottom. The further out on any axis you go, the stronger the emotion: so "extremely happy" would be in the top right hand corner of the graph.

This is the original piece: entertainer.mid

And here are the eight variants: e1 e2 e3 e4 e5 e6 e7 e8

Make a copy of the graph in your image editor of choice and for each variant, and for the original piece, please place a cross (and label it!) at the point you think best describes the emotion of the piece - whether it's happy or sad or angry etc., *not* how much you like this variation. All the crosses should be on the same graph, and the positions should all make sense relative to each other, so if you think one variant is very happy and then later find another variant is even happier, you might want to adjust your scale and move the first one closer to the centre.

Once you're done, return the completed image to me along with:

- Your name (so I can thank you on the report!)
- Your age
- Your level of musical experience (none / beginner / intermediate / advanced)

Thank you!

D.2 Experimental Values

This section details the 2DES coordinates given to ACRONYM for the two emotion recognition experiments, and the transformations calculated to achieve the desired emotion.

D.2.1 Emotion Recognition I

Variation	**e1**	**e2**	**e3**	**e4**
2DES coord	-0.25,0.22	-0.68,-0.83	0.33,0.35	0.3,-0.27
Mode	minor	minor	major	major
Tempo Factor	1.22	0.5833334	1.3533334	0.8666667
Articulation	1.0166667	1.7566667	0.66	0.98333335
Volume	1.0575329	0.4425007	1.2443216	0.9287987
Harmony	1.1722301	1.1674111	0.76348895	0.9052285

Variation	**e5**	**e6**	**e7**	**e8**
2DES coord	-0.77,0.76	0.77,-0.74	0.8,0.81	-0.27,-0.22
Mode	minor	major	major	minor
Tempo Factor	1.76	0.63	1.8133333	0.89
Articulation	1.0033333	0.9866667	0.19333333	1.2466667
Volume	1.2212181	0.7887819	1.5723522	0.83339083
Harmony	1.5407345	0.7699257	0.43155313	1.0911032

D.2.2 Emotion Recognition II

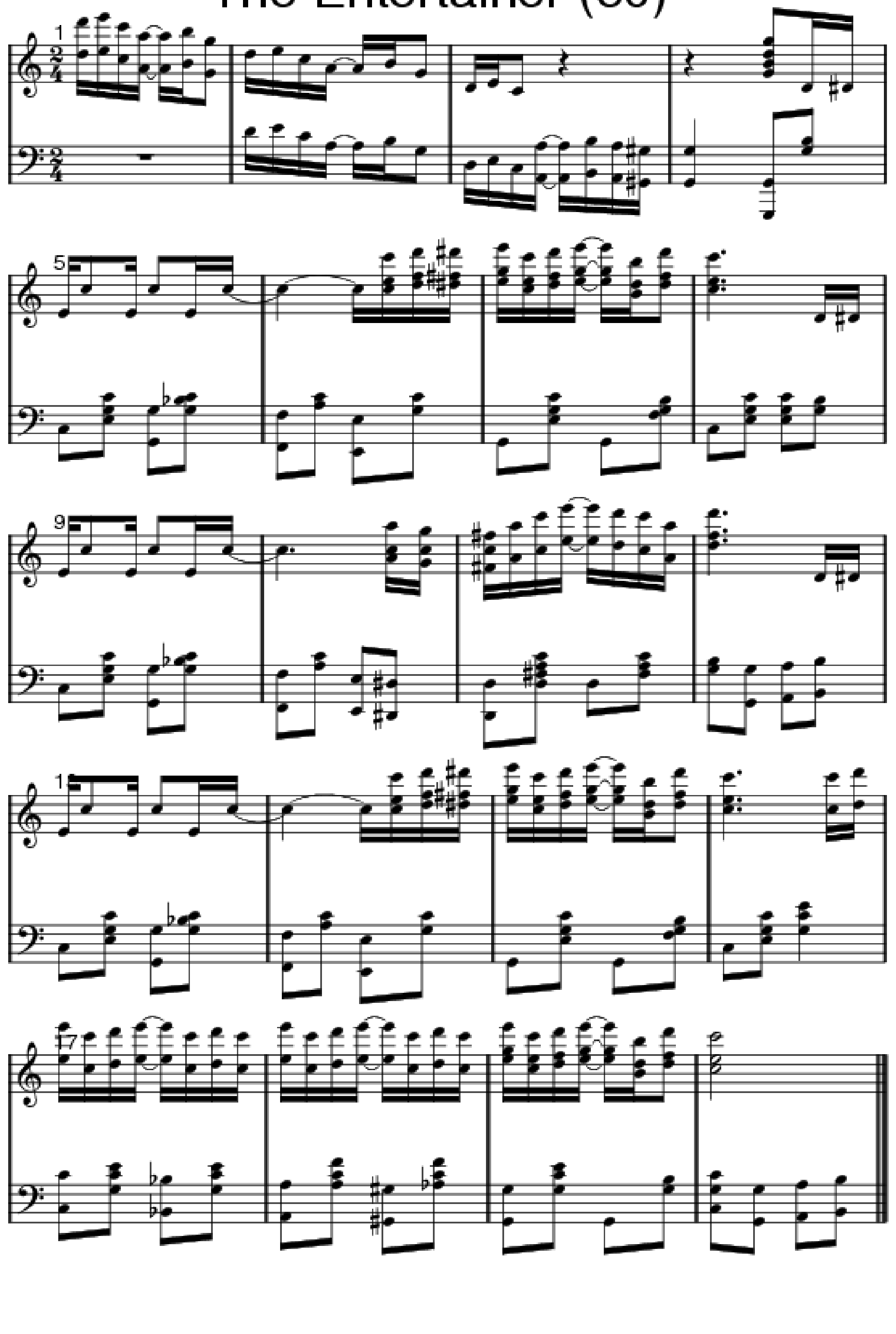
Variation	**s1**	**s2**	**s3**	**s4**
2DES coord	0.27,-0.22	0.75,0.78	-0.77,-0.79	-0.24,0.21
Mode	major	major	minor	minor
Tempo Factor	0.945	1.39	0.80333334	1.1066667
Articulation	0.979	0.31300002	0.81546444	1.0133333
Volume	0.98173916	1.2723198	1.2443216	1.0284805
Harmony	0.97805756	0.86628085	1.1118714	1.0820913

Variation	**s5**	**s6**	**s7**	**s8**
2DES coord	0.26,0.23	-0.79,0.77	0.73,-0.73	-0.27,-0.23
Mode	major	minor	major	minor
Tempo Factor	1.1133333	1.3833333	0.81666666	0.9433333
Articulation	0.781	1.01	1.0	1.2466667
Volume	1.0835905	1.1102047	0.92838955	0.94380146
Harmony	0.95576394	1.2760576	0.9463029	1.0431945

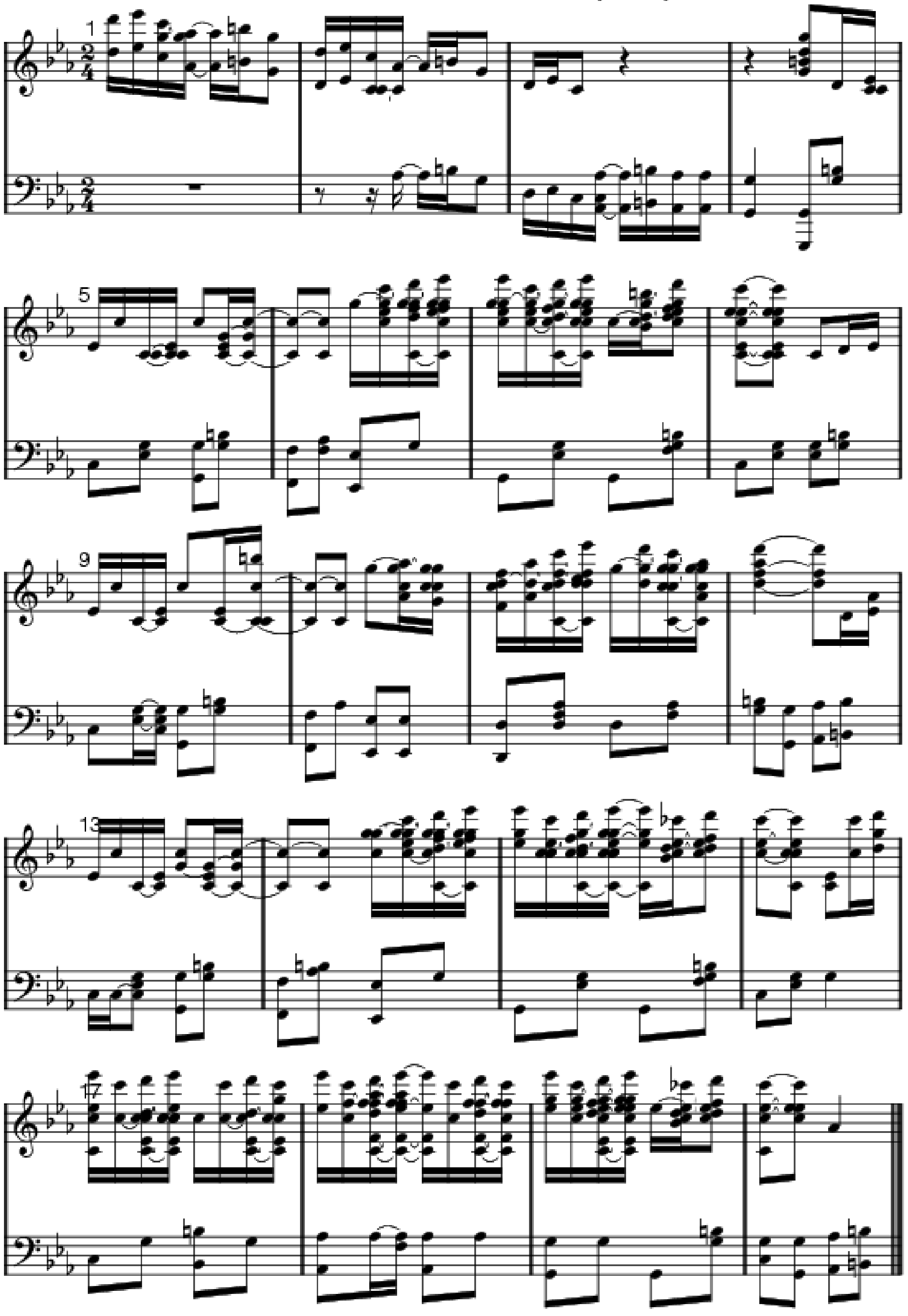
D.3 Sheet Music

Sheet music for the pieces used in the experiments is provided on the following pages. The scores were created using Cubase VST/32; note that grace notes in *Moments Musicaux No. 3* are not represented properly by Cubase.

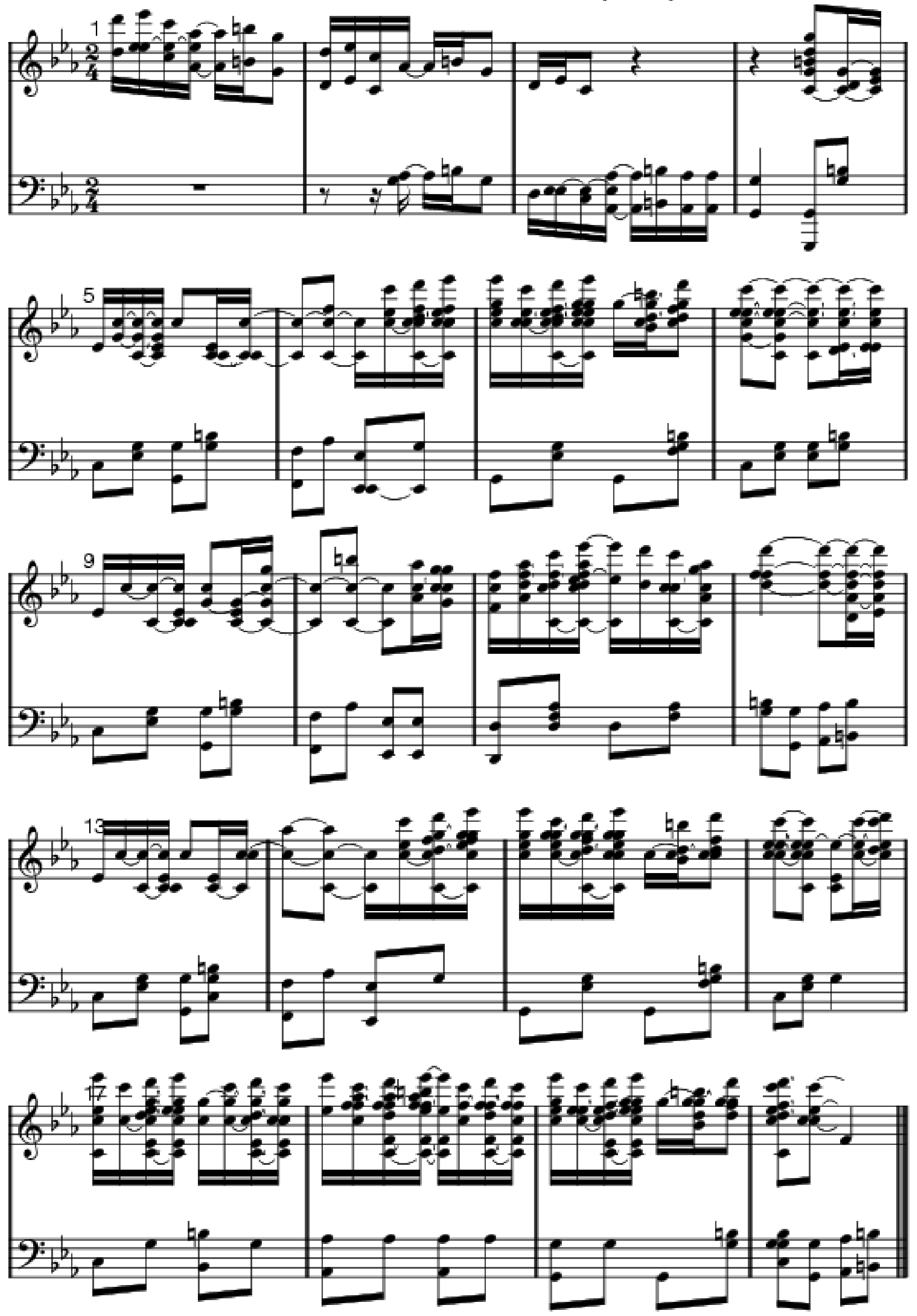
The Entertainer (e0)



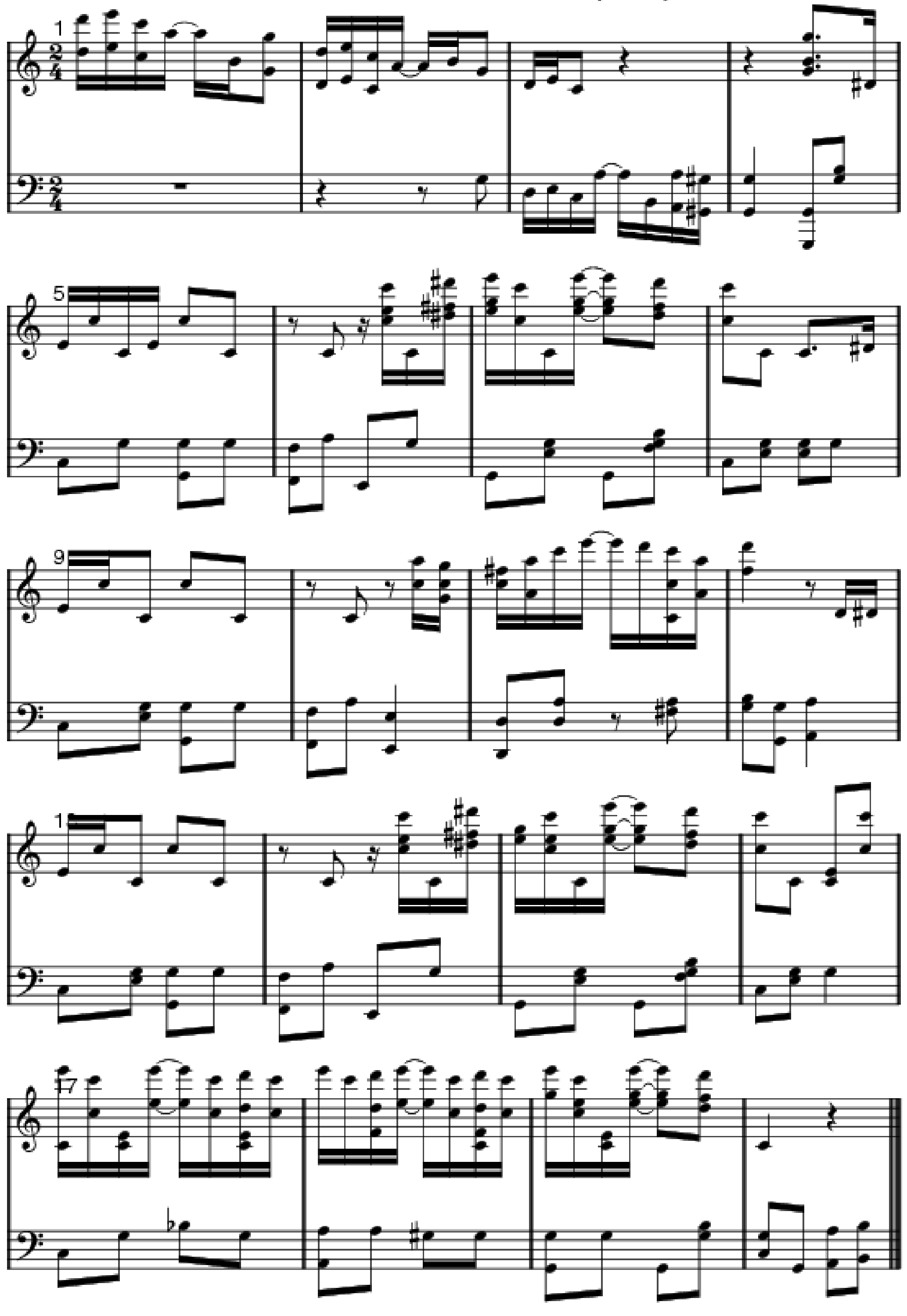
The Entertainer (e1)



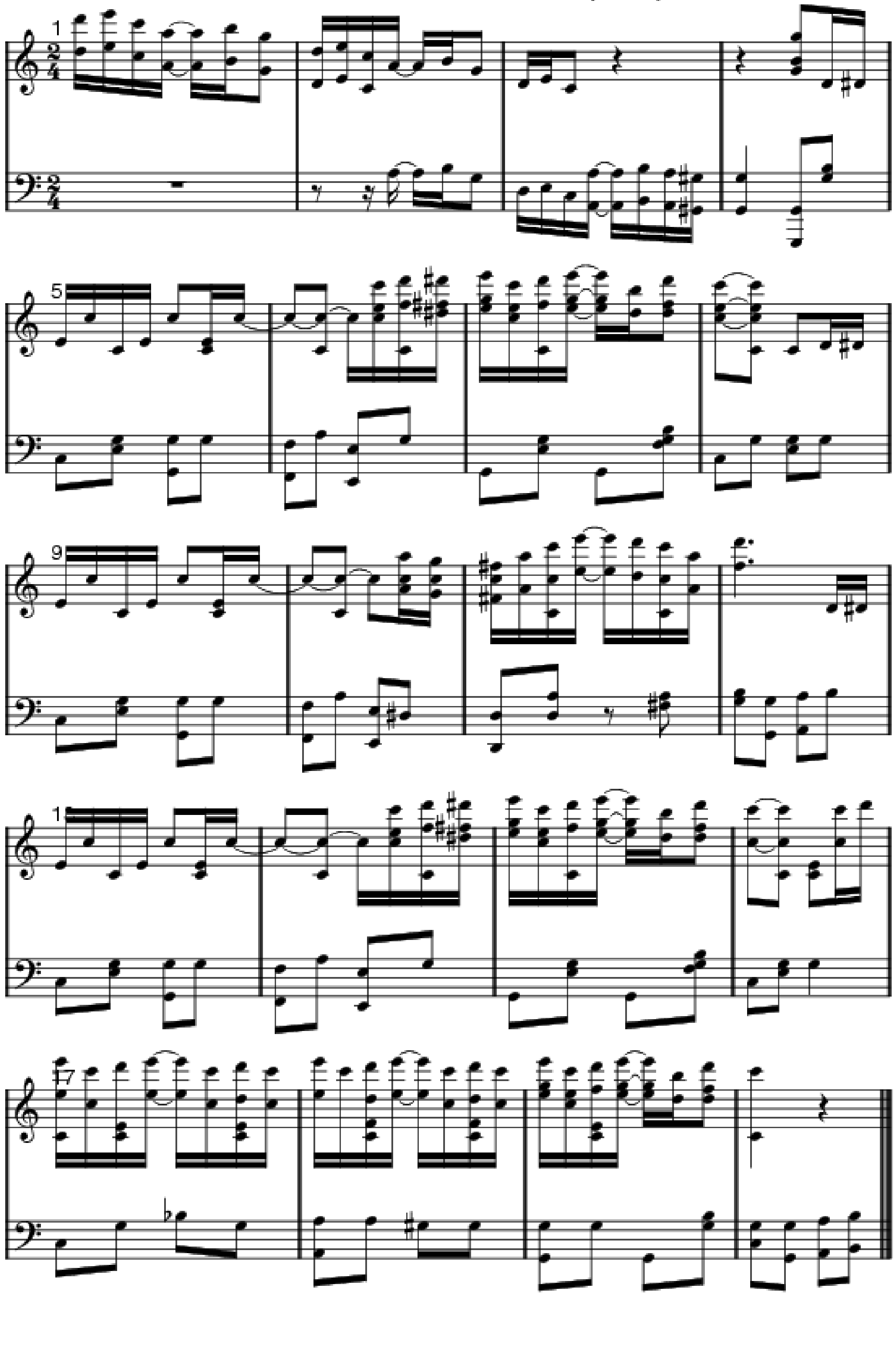
The Entertainer (e2)



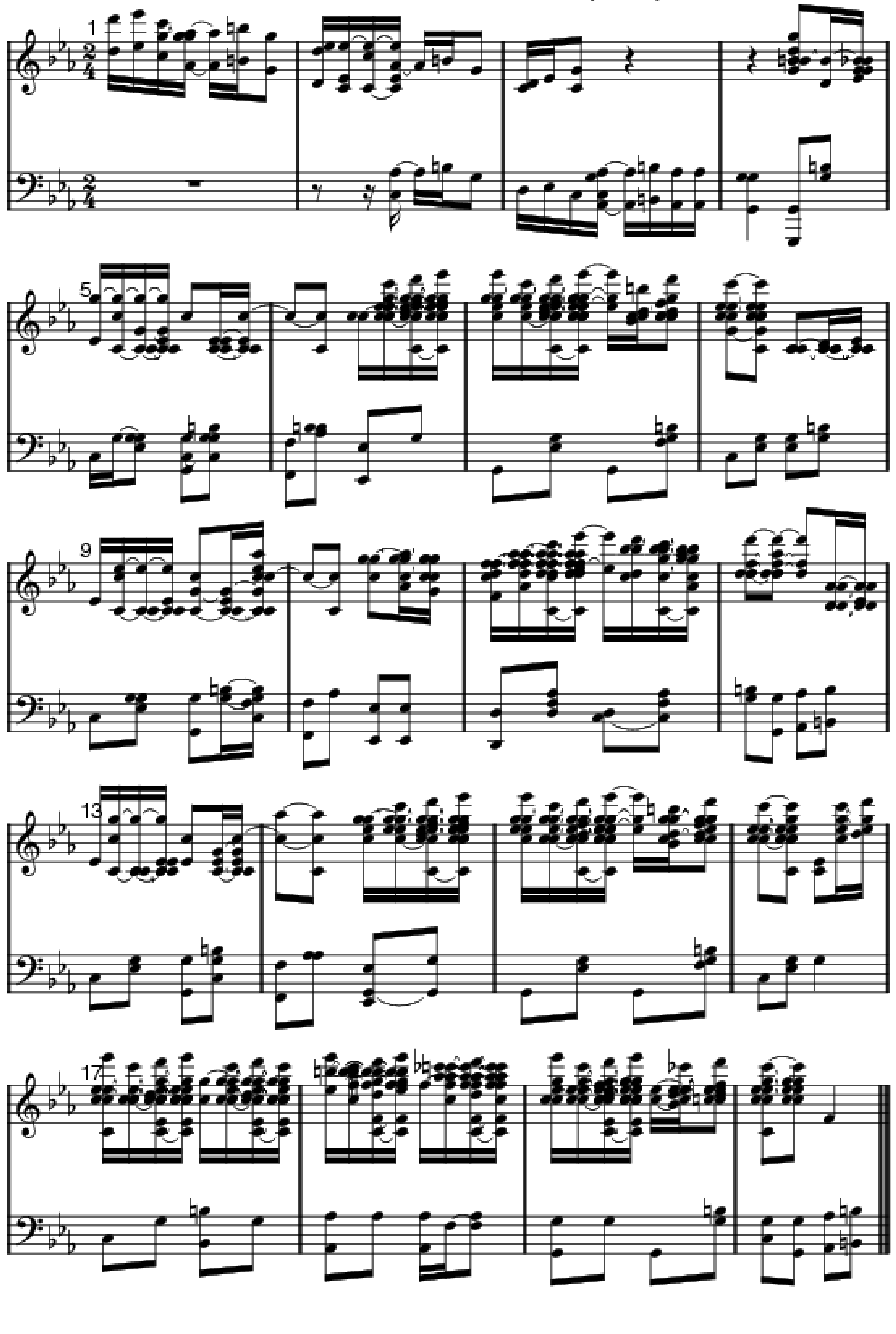
The Entertainer (e3)



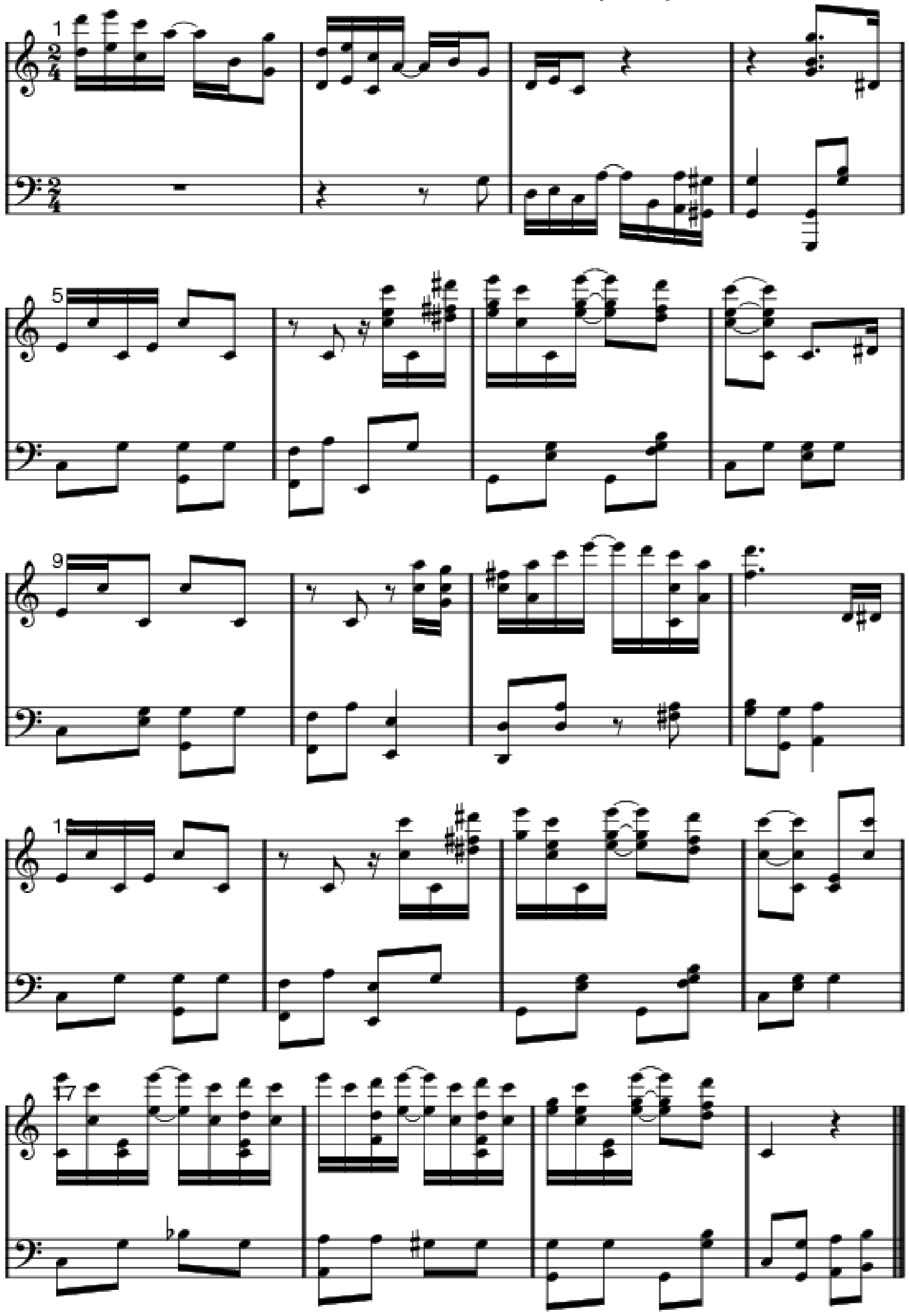
The Entertainer (e4)



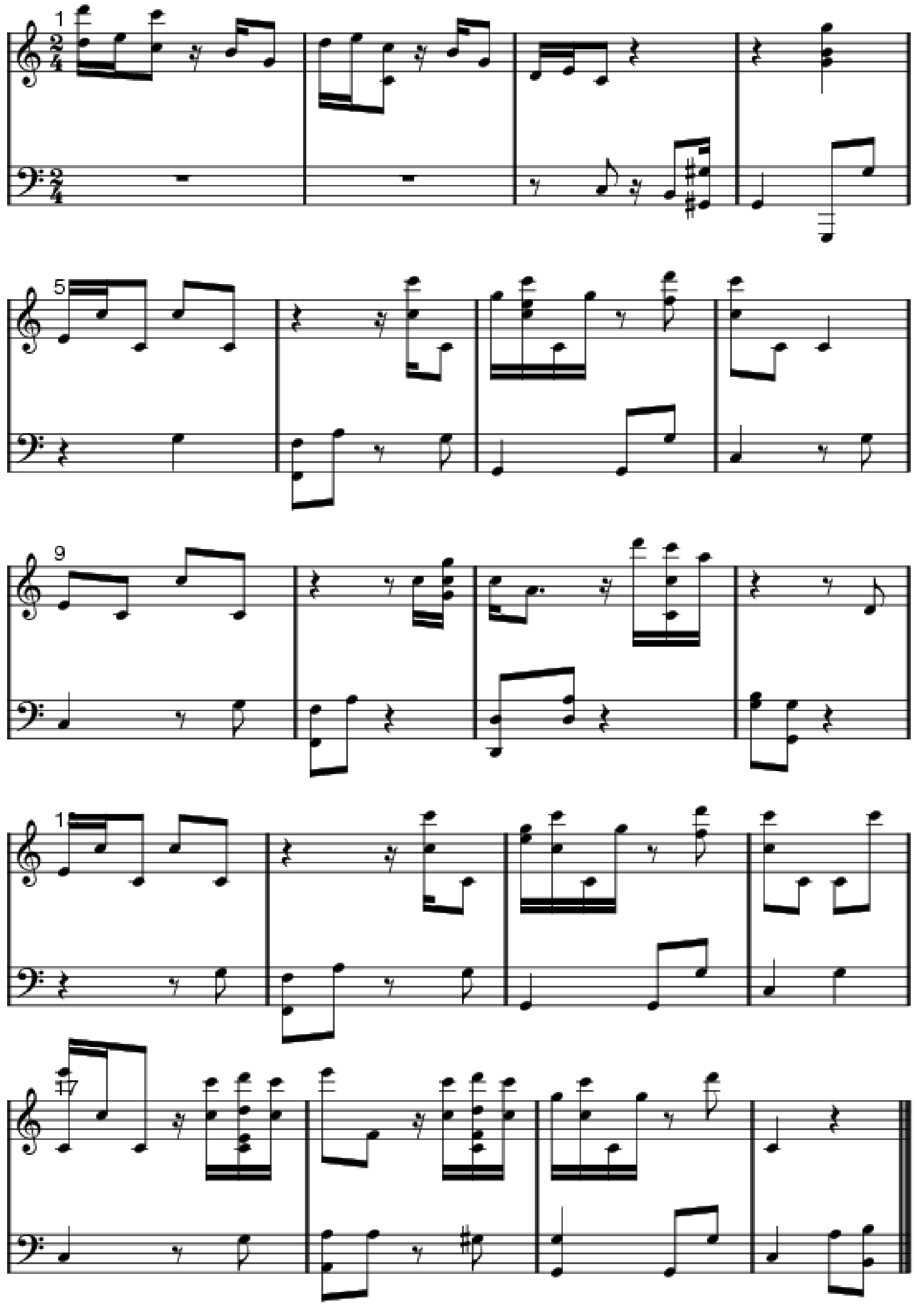
The Entertainer (e5)



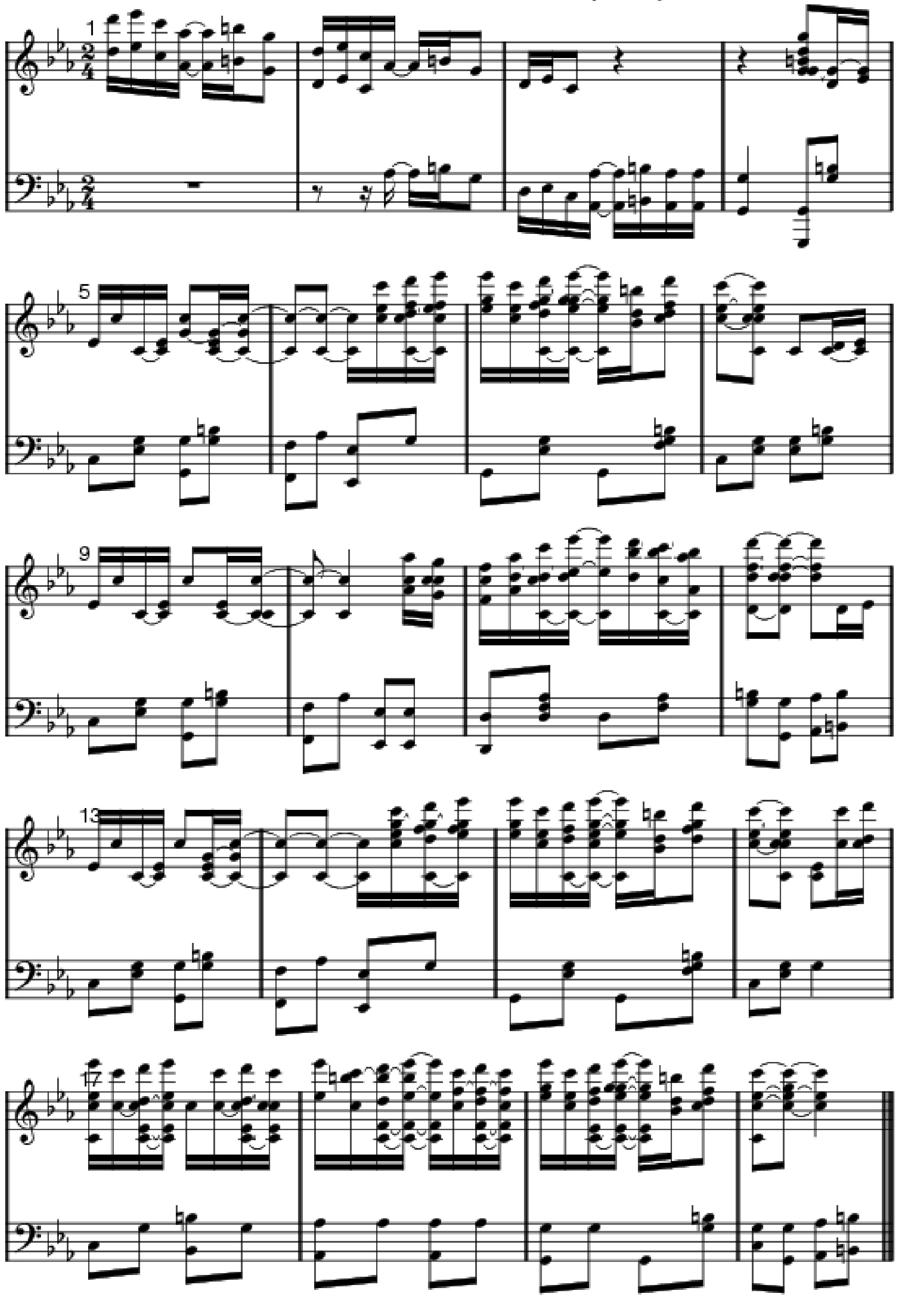
The Entertainer (e6)



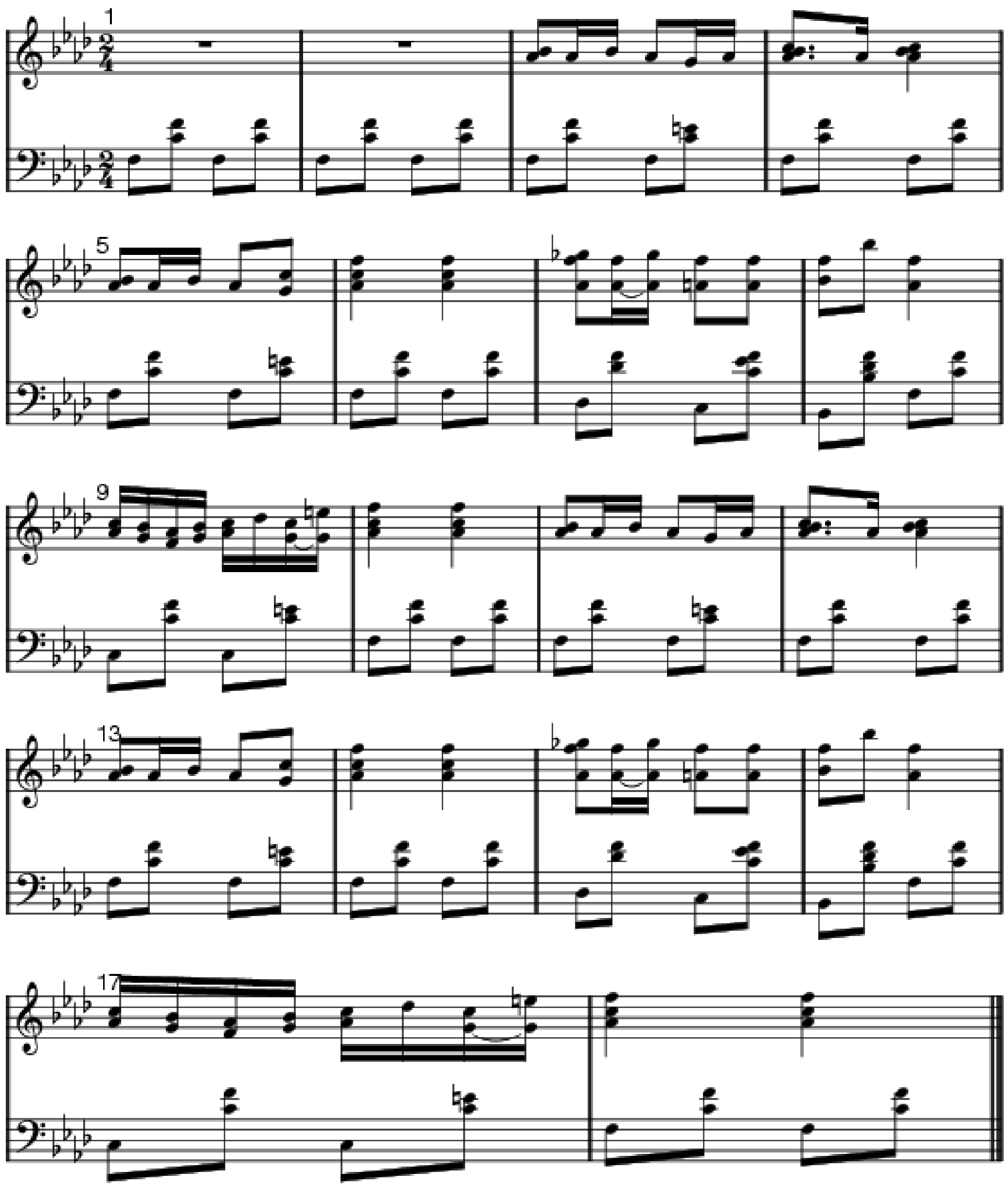
The Entertainer (e7)



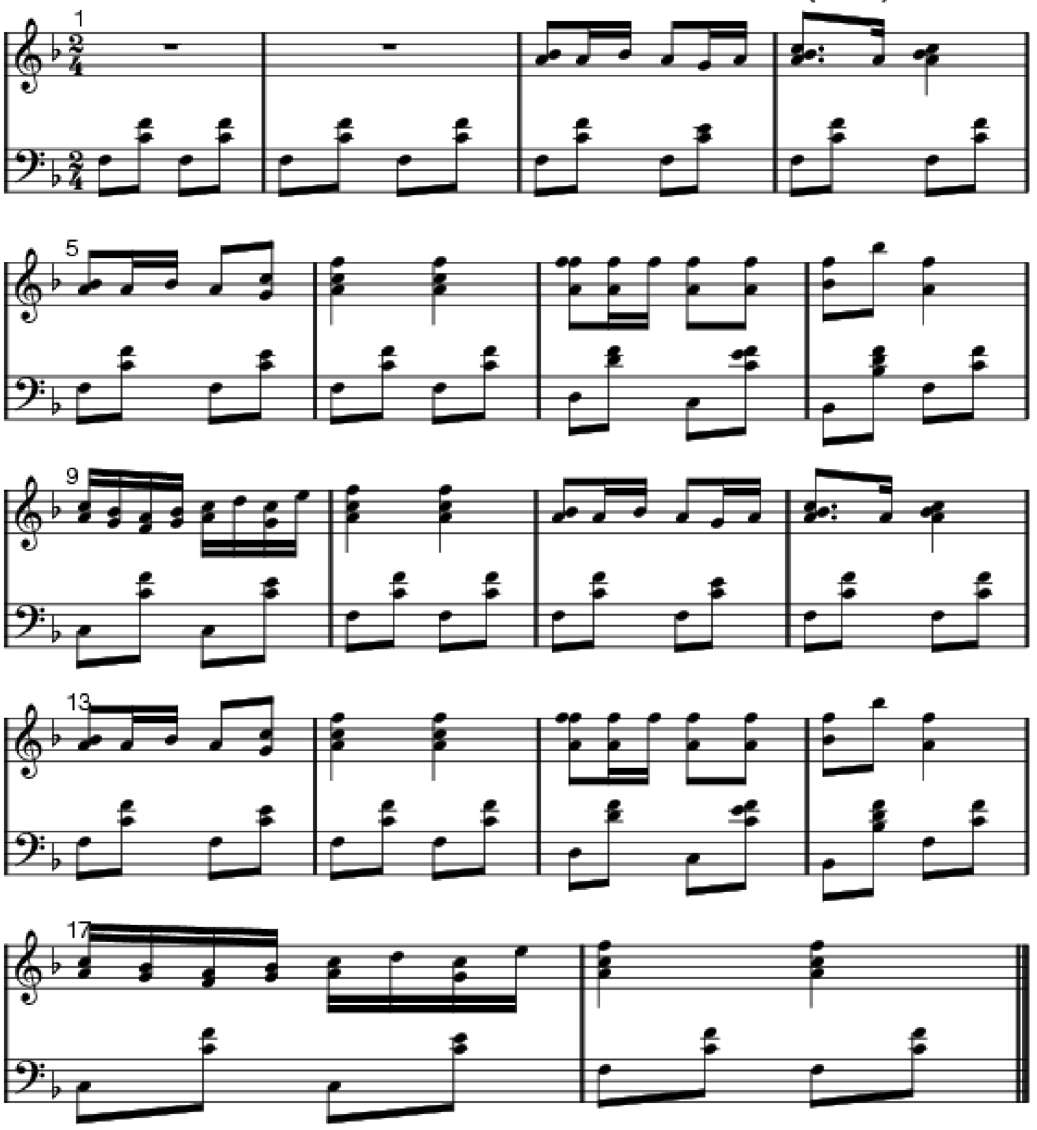
The Entertainer (e8)



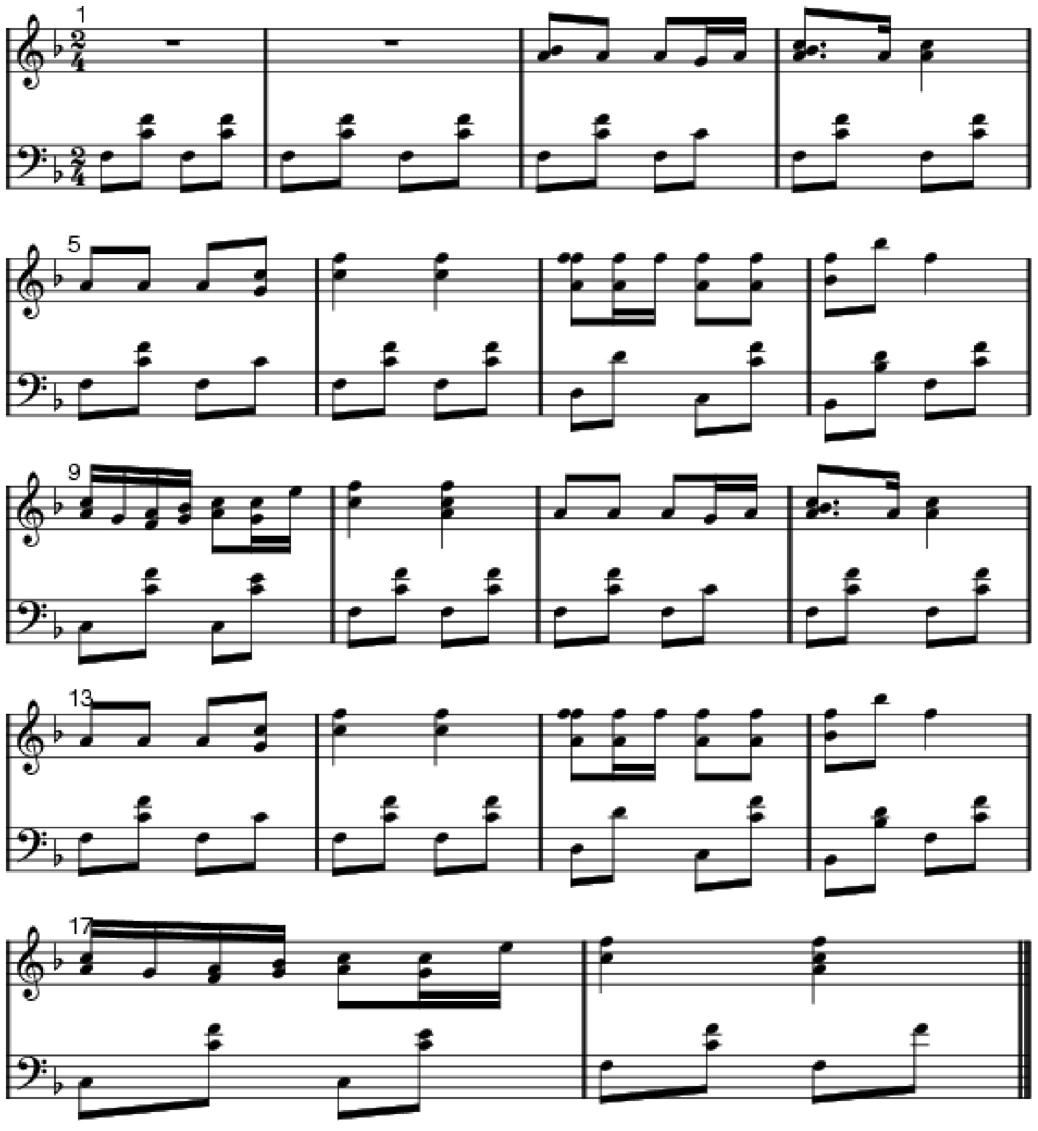
Moments Musicaux No. 3 (s0)



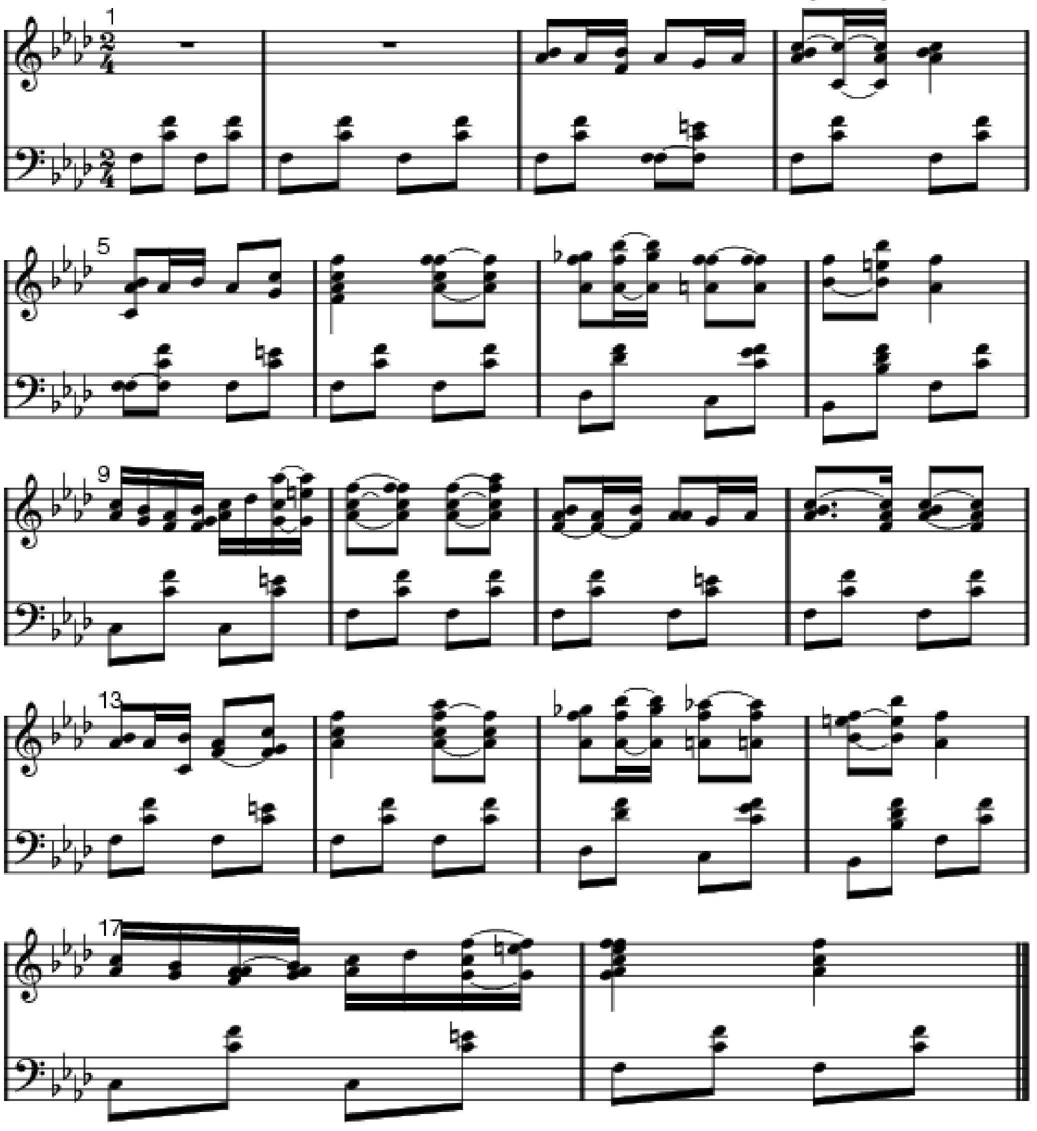
Moments Musicaux No. 3 (s1)



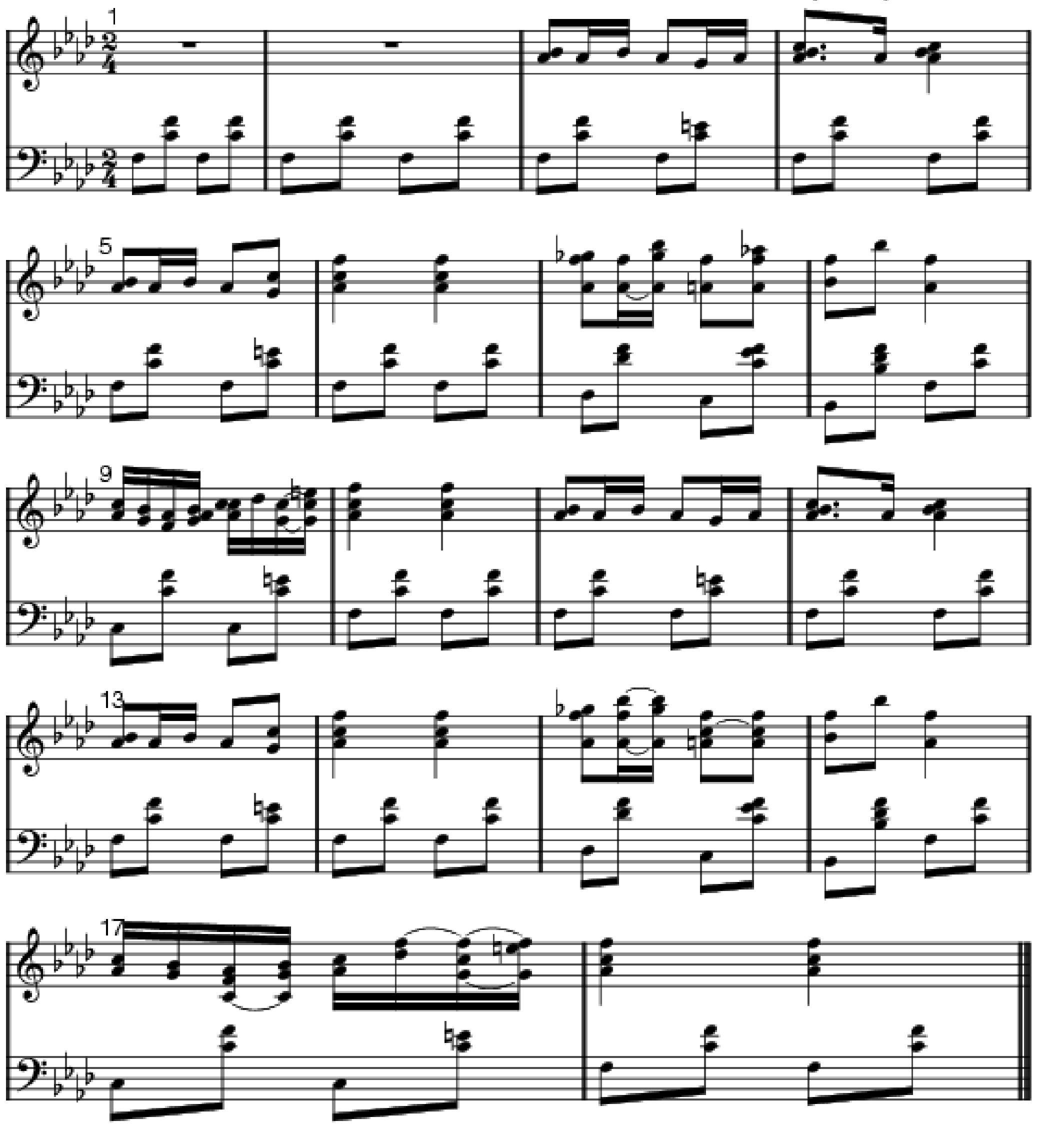
Moments Musicaux No. 3 (s2)



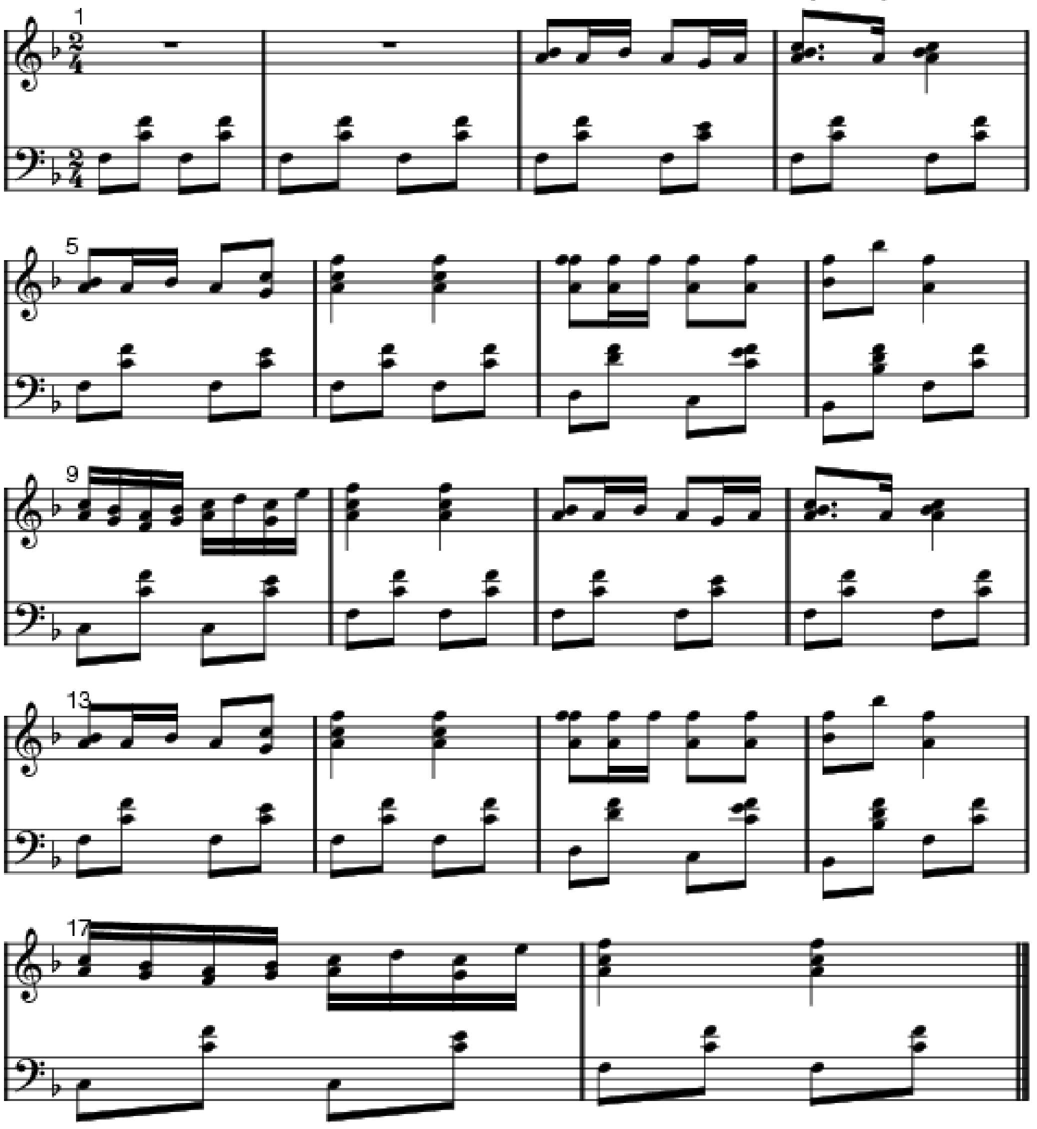
Moments Musicaux No. 3 (s3)



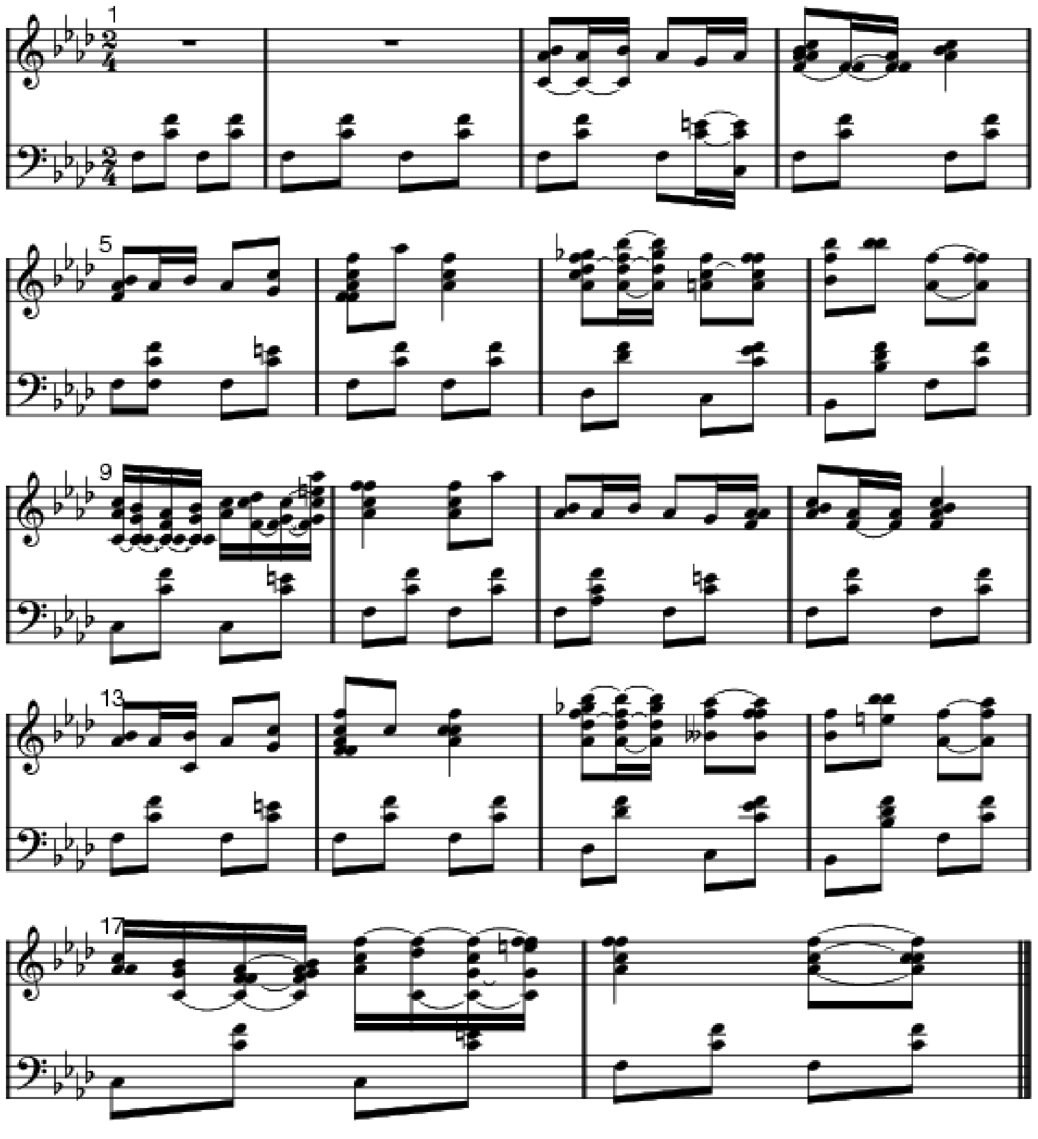
Moments Musicaux No. 3 (s4)



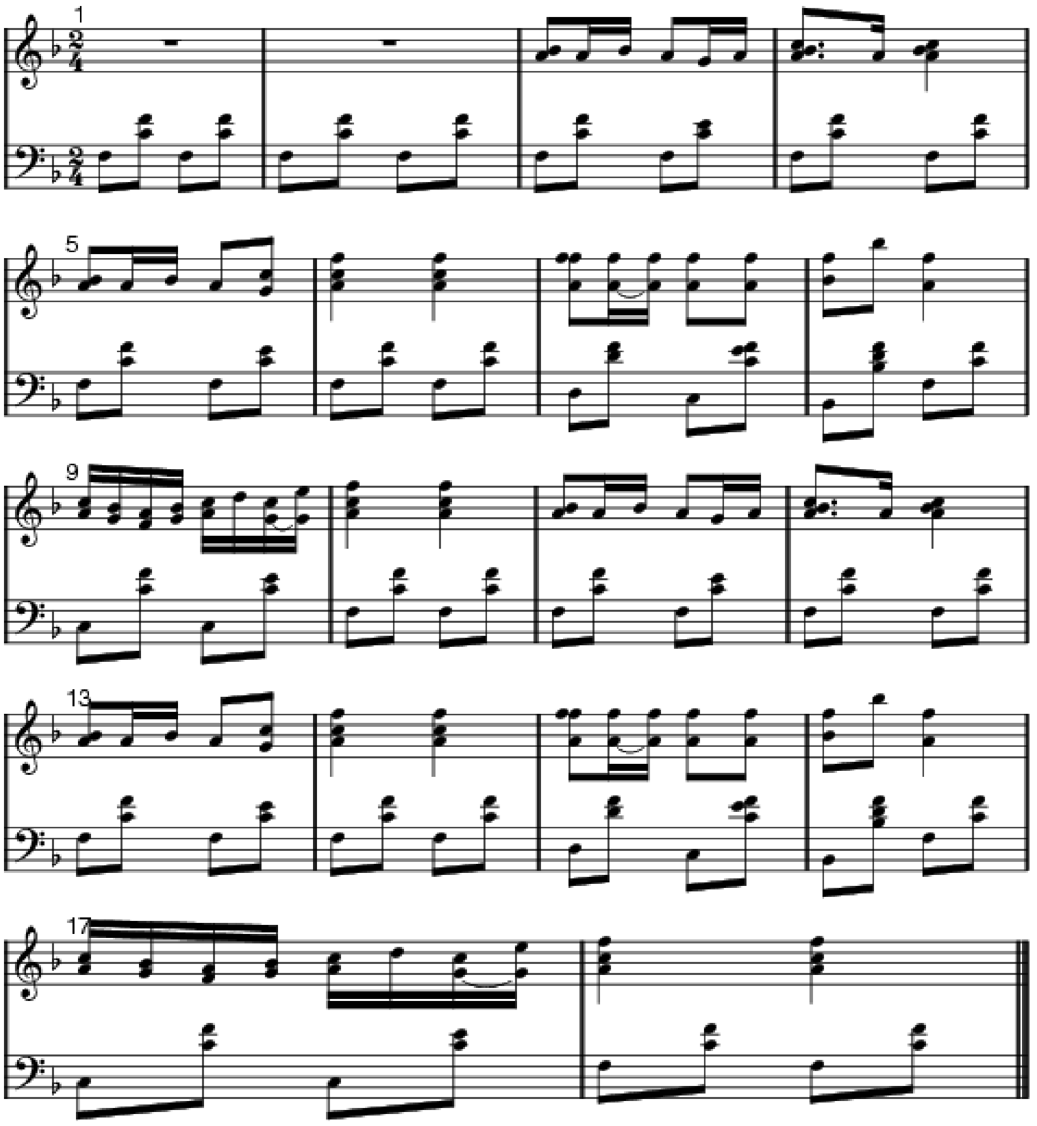
Moments Musicaux No. 3 (s5)



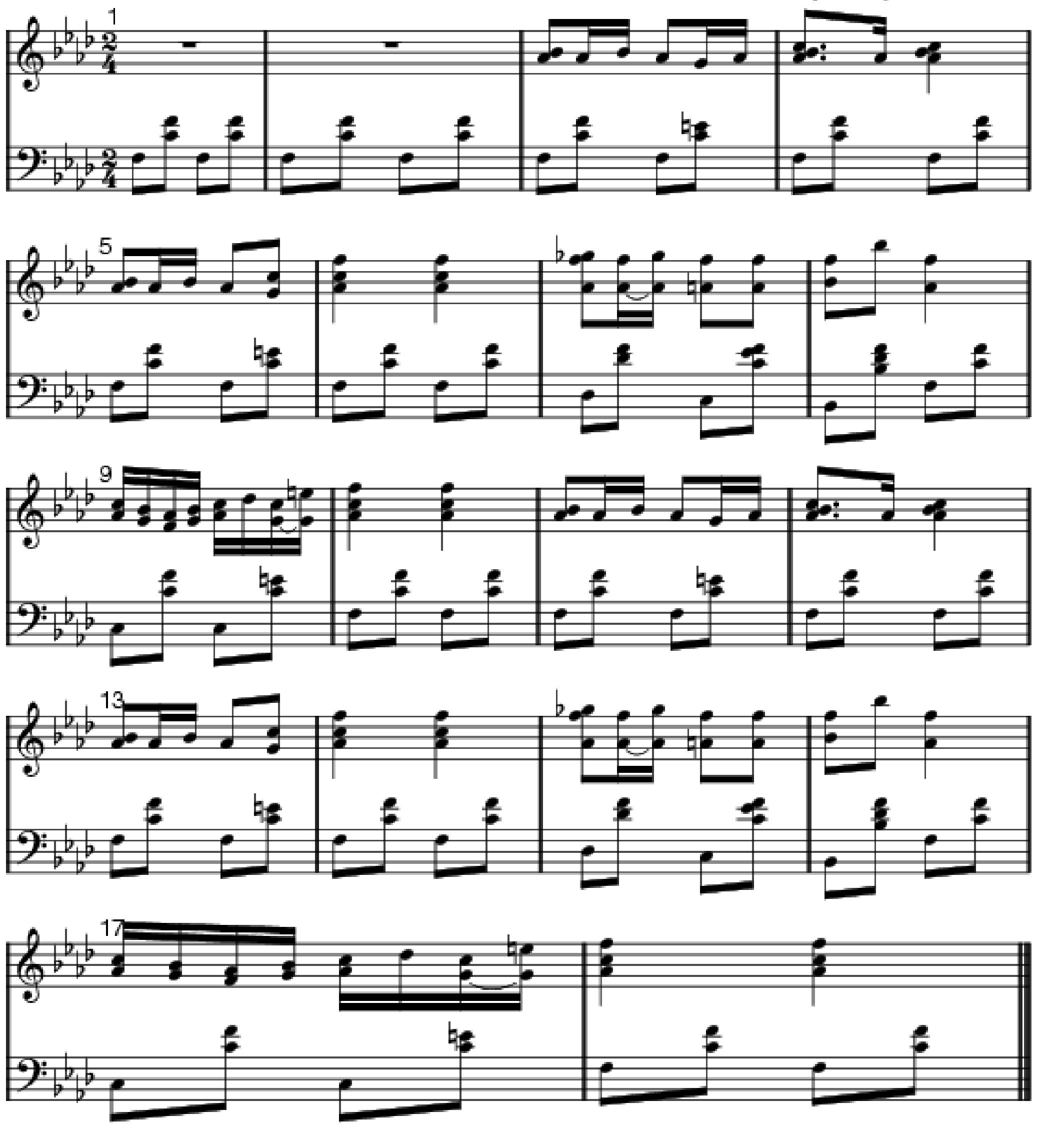
Moments Musicaux No. 3 (s6)



Moments Musicaux No. 3 (s7)



Moments Musicaux No. 3 (s8)



Bibliography

[1] 2K Boston / 2K Australia (2007). BioShock.

[2] Atari (1972). Pong.

[3] Atari (1979). Asteroids.

[4] Bally Midway (1983). Spy Hunter.

[5] Bresin, R. and Friberg, A. (2000). Emotional coloring of computer-controlled music performances. *Computer Music Journal*, 24(4):44–63.

[6] Capcom (2006). Phoenix Wright: Ace Attorney - Justice for All. The Phoenix Wright: Ace Attorney series was originally released in Japan for the Game Boy Advance under the title *Gyakuten Saiban*.

[7] Cinematronics (1983). Dragon's Lair.

[8] Croal, N. (2007). N'Gai Croal Vs. Roger Ebert Vs. Clive Barker on Whether Videogames Can Be (High) Art. Round 1–Fight! `http://blog.newsweek.com/blogs/levelup/archive/2007/07/30/croal-vs-ebert-vs-barker-on-whether-videogames-can-be-high-art-round-1.aspx`.

[9] Data Age (1982). Journey Escape.

[10] Douglas, A. (1952). OXO. Illustration for PhD thesis, University of Cambridge.

[11] Enix (1991). ActRaiser.

[12] Farnsworth, P. R. (1954). A study of the Hevner adjective list. *Journal of Aesthetics and Art Criticism*, 13:97–103.

[13] Farnsworth, P. R. (1969). *The social psychology of music*. Iowa State University Press, Ames, Iowa, 2nd edition.

[14] Gabrielsson, A. (1994). Intention and emotional expression in music performance. In Friberg, A., Iwarsson, J., Jansson, E., and Sundberg, J., editors, *Proceedings of the Stockholm Music Acoustics Conference*, pages 108–111, Stockholm. Royal Swedish Academy of Music.

[15] Gabrielsson, A. (1995). *Music and the Mind Machine: The Psychophysiology and the Psychopathology of the Sense of Music*, pages 35–47. Springer Verlag, Berlin.

[16] Golden T Studios (2005). Golden T Game Engine. `http://www.goldenstudios.or.id/products/GTGE/`.

[17] Harmonix Music Systems (2005). Guitar Hero.

[18] Harmonix Music Systems (2007). Rock Band.

[19] Hevner, K. (1935). The affective character of the major and minor modes in music. *The American Journal of Psychology*, 47(1):103–118.

[20] id Software (1996). Quake.

[21] Io Interactive (2000). Hitman.

[22] James, W. (1884). What is an emotion? *Mind*, 9:188–205.

[23] Juslin, P. (1997a). Emotional communication in music performance: A functionalist perspective and some data. *Music Perception*, 14(4):383–418.

[24] Juslin, P. (1997b). Perceived emotional expression in synthesized performances of a short melody: Capturing the listener's judgment policy. *Musicae Scientiae*, 1(2):225–256.

[25] Konami (1998). Dance Dance Revolution.

[26] Konami (2003). Karaoke Revolution.

[27] Land, M. Z. and McConnell, P. N. (1991). Method and apparatus for dynamically composing music and sound effects using a computer entertainment system. U.S. Patent #5,315,057.

[28] Lionhead Studios (2008). Fable 2.

[29] Livingstone, S. R. and Brown, A. (2005). Dynamic response: Real-time adaptation for music emotion. In *Australasian Conference on Interactive Entertainment.*

[30] Livingstone, S. R., Brown, A. R., and Muhlberger, R. (2005). Influencing the perceived emotions of music with intent. In Innocent, T., editor, *Third Iteration*, pages 161–170.

[31] LucasArts (1990). The Secret of Monkey Island.

[32] LucasArts (1991). Monkey Island 2: LeChuck's Revenge.

[33] Mattel Electronics (1979). Major League Baseball.

[34] Midway Mfg. Co. (1975). Gun Fight. This game was imported from Taito in Japan and converted by Midway to use the Intel 8080 processor.

[35] Namco (1980). Pac-Man.

[36] Nintendo (1985). Super Mario Bros.

[37] Nintendo (1987a). Final Fantasy.

[38] Nintendo (1987b). The Legend of Zelda.

[39] Nintendo (1998). The Legend of Zelda: Ocarina of Time.

[40] Nintendo (2001). Super Smash Bros. Melee.

[41] Nintendo (2007). Super Mario Galaxy.

[42] Pajitnov, A. (1985). Tetris.

[43] Plutchik, R. (1980). *Emotion: Theories, research, and experience: Vol. 1. Theories of emotion*, pages 3–33. Academic Press, New York.

[44] Psygnosis (1996). Wipeout XL. This game was also known as Wipeout 2097.

[45] Schubert, E. (1996a). Continuous response to music using a two dimensional emotion space. In Pennycook, B. and Costa-Giomi, E., editors, *Proceedings of the 4th International Conference of Music Perception and Cognition*, pages 263–268, Montreal. McGill University.

[46] Schubert, E. (1996b). Measurement and time series analysis of emotion in music. Master's thesis, University of New South Wales, New South Wales, Australia.

[47] Sega (1989). Michael Jackson's Moonwalker.

[48] Sega (1991). Streets of Rage.

[49] Sega (1993). Sonic CD.

[50] Sega (2001). REZ.

[51] Sorensen, A. and Brown, A. (2007). jMusic. `http://jmusic.ci.qut.edu.au/`.

[52] Taito (1978). Space Invaders.

[53] Valve Corporation (2007). Portal.

[54] Whissell, C., Fournier, M., Pelland, R., Weir, D., and Makarec, K. (1986). A dictionary of affect in language: IV. reliability, validity and applications. *Perceptual and Motor Skills*, 62:875–888.