barelyMusician: An Adaptive Music Engine For Video Games

Conference Paper · February 2015

CITATIONS
2

READS
142

3 authors:

Alper Gungormusler
Google Inc.
2 PUBLICATIONS **2** CITATIONS
SEE PROFILE

Natasa Paterson
Trinity College Dublin
8 PUBLICATIONS **80** CITATIONS
SEE PROFILE

Mads Haahr
Trinity College Dublin
70 PUBLICATIONS **1,881** CITATIONS
SEE PROFILE

barelyMusician: An Adaptive Music Engine For Video Games

Alper Gungormusler[1], Natasa Paterson-Paulberg[1], and Mads Haahr[1]

[1]*Trinity College Dublin, College Green, Dublin 2, Ireland*

Correspondence should be addressed to Alper Gungormusler (gungorma@tcd.ie)

ABSTRACT

Aural feedback plays a crucial part in the field of interactive entertainment when delivering the desired experience to the audience, particularly in video games. It is, however, not yet fully explored in the industry, specifically in terms of interactivity of musical elements. In this paper, we present *barelyMusician*, an extensible adaptive music engine that offers a new set of features for interactive music generation and performance for highly interactive applications. *barelyMusician* is a comprehensive music composition tool, capable of generating and manipulating audio samples and musical parameters in real-time in order to create smooth transitions between musical patterns that portray varying emotional states and moods that may be evident during gameplay. The paper presents the underlying approach, features and user interface, as well as a preliminary evaluation through demonstrators.

1. INTRODUCTION

There has been a tremendous developmental curve in regards to the technological aspects of the interactive entertainment field in the last two decades—specifically in the video game industry. While the main focus for many years has been on improving visual fidelity, the maturation of graphics technologies has led to developers focusing on other game parameters in order to further engage players in the game experience. In particular, audio is known to have a dramatic effect on player immersion [1].

Visually, video games have traditionally included non-interactive elements, such as fixed-camera cut-scenes, but these are increasingly being replaced with interactive sequences. Similarly, game music, not least in high-budget mainstream products [2], has been modelled on non-interactive (or linear) film music. Even though many rich and sophisticated scores have been composed, they tend to lack the main feature required; interactivity. Unlike movies, video games as interactive applications rarely follow a linear path, and more importantly offer a great range of repeatability which makes dynamism—hence, adaptivity—a must to have in terms of the audio content, as it is in other parts of such applications. While a few examples exist in the industry which try to achieve good adaptivity in audio, the issue has not been solved yet to an acceptable level particularly considering the practical aspects. More specifically, existing technologies in the field, labeled as dynamic, generally rely on *the vertical approach* [3]—layering and branching the composition into segments. Thus, the resulting output can be varied in an automated manner by certain event triggering approaches. The idea is, however, still based on pre-recorded audio loops which do not allow full dynamic flexibility after the offline creation process. Moreover, such approaches require a considerable amount of resources and dependencies mainly due to the manual authoring of the outcome.

While there is increasing interest to produce adaptive and responsive aural feedback [4] as well as in interactive sound synthesis in academic research and industry applications, achieving adaptive music compositions in real-time remains an unsolved problem. There have been several attempts in the industry to take the approach one step further by introducing certain generative algorithms in order to create the musical elements procedurally [3, 5]. However, these remained as specific solutions exclusive only for those specific projects. Having considered this, there is no generic approach, i.e., an all purpose solution with such capabilities in the field.

To address the issues and the potential need stated above, this paper proposes a practical approach for the development of a middleware tool, *an adaptive music engine*, which is capable of autonomously generating and manipulating musical structures in real-time to be used by

developers as well as designers. The proposed approach treats the problem in an interactive manner, providing a bridge between the low-level properties of a musical sound and the high-level abstractions of a musical composition which are meant to be significant to the user.

2. BACKGROUND & RELATED WORK

Music as an art form has limitless possibilities for the combination and permutation of different soundscapes to create a compositional piece. Nevertheless, when considering the music theory, creating a musical composition is a highly structured and rather rule-based process particularly when observing Western music in the 18th and 19th centuries, or even more recently in such genres as pop or jazz music. These types of compositions are generally defined as a part of tonal music. That being said, tonality in music could be described as a system of musical patterns—an organisation of melodic intervals and chords arranged in a hierarchical way to be perceived by the listener as tonal music [6].

While it overlooks the artistic means of creation, achieving tonality in a musical piece is arguably straightforward, if one could treat the musical properties in a systematic way. Thus, it is fairly possible to produce a musically recognisable sequence of soundscapes just by following certain rules in the right order. It is expected that this approach does not give the most fascinating and/or inspiring outcome for a composer, but nonetheless, the result will be an adequate musical piece that sounds "good" to a typical Western listener [7]. The overall structure of the composition should also be considered at a macro level in order to provide an interesting and meaningful outcome as a complete compositional piece. Having considered this, a musical composition could be thought of as different levels of organisation, similar to a language form in linguistics [8]. Sectional form is commonly used in music theory to arrange and describe a music composition in that manner. In sectional form, the musical piece is made of a sequence of *sections*, which are analogous to *paragraphs* in linguistics. Sections are often notated by single letters such as *A* and *B* to make them easier to read. Unlike paragraphs, sections might occur more than once in a musical piece. In fact, it generally is not only preferred but also essential to communicate the musical message, i.e., the repetition of sections can contribute to develop patterns of recognition which can create perceived meaning and an emotional response for the listener [9]. In popular music, sections are traditionally referred under specific names (rather than single letters) such as *verse*, *chorus* and *bridge*.

Furthermore, each section in a piece consists of a certain number of *bars*—also known as *measures*—to build up the song structure. Likewise, each bar is made of a certain number of *beats* as punctual points, which are traditionally notated with time signatures on the musical staff. For instance, the most common form, 4/4, claims that each bar in the piece has four beats (numerator) which are all quarter notes (denominator).

As a structural approach, such properties could be made by certain algorithmic techniques used in tonal Western music to automate the entire behaviour providing an artificially intelligent system. One early example of such a system is the *Iliac Suite* (1956) by Hiller and Isaacson [10]. This computer generated piece used the musical rules of Baroque and twelve tone music to produce a notated score which was then performed by a an acoustic string quartet. Recently, this approach has been adopted for procedural music composition through algorithmic techniques, such as Markov models, generative grammars and genetic algorithms [11]. There also exist a number of software solutions which simplify the developmental process by providing certain low-level architectures to be used in the sound and music generation process.

These algorithmic techniques can be used to create adaptive music generation systems. In fact, the term "adaptive music", which refers to the use of interactive elements in real-time, has been introduced in the field long ago including some successful examples in major video games such as *Monkey Island 2: LeChuck's Revenge* (1991) by LucasArts [12]. This particular game features an interactive music streaming engine to seamlessly synchronise the background music with the visual (on-screen) events in the game. However, the idea somewhat failed to develop further due to—arguably—two major reasons [13]. Firstly, there was low interest by both developers and consumers in the area of research because of the great focus on the technology behind the visual aspects such as graphics and physics systems. Secondly, despite their significant power, improvements and enhancements created for the quality of digitized audio data in computers actually resulted in a negative impact on the development, particularly when the industry shifted from basic MIDI representations to streamed—lossless—audio data. While the quality of the output increased dramatically, the huge amounts of streamed raw data could not be modified or manipulated efficiently in real-time with

existing computational technologies. Hence, it was not a desirable choice (nor a priority), in general, to spend the limited resources to sophisticate audio elements in an application early on. It is only recently that there has been notable research done in the field [14].

One such example is AMEE [15, 16]—a real-time music generation system which features an adaptive methodology in accordance with the desired emotional characteristics of a musical piece. The system is able to expressively generate musical compositions in real-time with respect to the selected properties—i.e., the perceived moods—in an interactive manner. It furthermore features an application programming interface written in Java for external usage. However, one major drawback of the approach is the lack of smooth transitions between the different selections of musical pieces, which is said to be left as future work. While the proposed version offers an interactive way to modify the musical parameters in real-time, the change in the output only occurs after a period of time, more specifically, after the generated musical block completes its playback. Additionally, its quality of audio is currently limited to MIDI files, which makes the system somewhat inadequate for use in practical applications. Besides, even if the MIDI information is somehow converted to rich audio samples in real-time, the approach lacks low-level soundscape manipulation due to the absence of essential information such as audio envelope properties.

To take the level of interactivity one step further, musical parameters could be examined and treated in a specific way to transform the musical pieces dramatically in real-time. Livingstone et al. [19, 20] and Friberg et al. [21] rather focus on the affective transformation of music in order to create smooth transitions between the different pieces in an effective way. They successfully transform the musical piece in real-time by using certain mapping methodologies between the perceived moods and musical parameters. A rule-based fitting algorithm is used in order to "fix" the musical properties (mainly the pitches of notes, hence the harmony), to a desired format which is determined by the user. In return, they both rely on either pre-existing audio tracks or pre-generated musical patterns to produce the output. Thus, they lack the ability to provide unique music generation in real-time.

In the game *Spore* (2008) by Maxis, most of the audio content is generated in real-time using mainly two dimensional cellular automata [17]. The concept is inspired from the unique yet sophisticated patterns which could be found in Conway's *Game of Life* (1970) experiments [18]. While a significant amount of the musical parameters are pre-defined, hence static, the resulting compositions could arguably be perceived as much more compelling than the other works reviewed in this section. The main reason is because of the use of real audio samples in order to offer a compatible audio output that a typical video game player in the industry is used to. On the other hand, the system is designed specifically for this particular game only. Therefore, generalisation of the architecture does not seem to be possible for later applications.

The most recent work currently in the field is *AUD.js* by Adam et al. [22]. The project is developed in Javascript and focuses on a smooth adaptation of the musical pieces in real-time. Based on Livingstone's approach [19], the system features two high-level parameters, namely "energy" and "stress", to be controlled by the user and is capable of making use of these values immediately in the generation process. It features a small-scale library of musical patterns to be selected on the fly. After the pattern selection, the output is generated by relatively primitive audio synthesis methods such as creating sine waveforms with the specified pitch and volume. Hence, it somewhat fails to produce sufficiently complex soundscapes as an output. Moreover, as in the previous examples, the system is not capable of generating the composition in real-time, which restricts its potential for wider usage.

In conclusion, the recent works tend to have a trade-off between the music generation and affective transition capabilities. Moreover, the final output is often not as compelling as the other examples using the traditional approaches in the industry. In fact, they rarely focus on providing more sophisticated methodologies in terms of audio production, or even to support multiple instruments in their systems. In fairness, as academic works, those issues are usually claimed as a future work to be augmented later on. Nonetheless, there exist no examples in the field that seem to progress the research further and address the given limitations.

3. DESIGN & IMPLEMENTATION

A hybrid approach was proposed for the design of *barelyMusician*, which combines real-time music generation and affective musical mood transition capabilities into a single yet powerful framework in order to take previous examples of adaptive music systems one step further. The approach focuses on the practical aspects

of the composition and transformation techniques and technologies. Hence, the design goal was to develop a generic comprehensive framework that abstracted the low-level musical properties and structures. This enables the system to be used intuitively in creating generative dynamic music without the need for any advanced knowledge of music theory. In addition, the system was designed to provide an interactive way to modify the high-level properties of the music composition process, such as the transitions between various musical moods, at runtime simply by triggering the corresponding functions respectively through an application programming interface and/or a graphical user interface.

3.1. Main Architecture

The main architecture of the system is loosely based on Hoeberechts et al.'s pipeline architecture for real-time music production [23]. The components of the architecture is designed in a way that reflects a typical real-life musical performance in order to divide the workload in a meaningful manner with respect to the end-user. As shown in Figure 1, *Musician* is the main component of the architecture which is responsible for the management and the communication between the other components in the engine. *Sequencer* serves as the main clock of the engine, i.e., it sends relevant signals to *Musician* whenever an audio event occurs. Audio events are designed hierarchically in terms of common sectional forms as described previously. That being said, *Sequencer* follows a quantised approach in which it keeps track of highly grained *audio pulses*[1] to occur in the audio sampling rate. Counting the pulses by a phasor [24], it sequences the beats, bars and sections respectively.

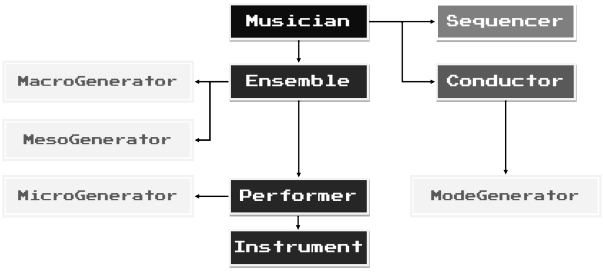

Fig. 1: Main components and dependencies.

In each audio event, *Musician* passes that information to the *Ensemble* component, i.e., to the orchestra. Ensemble creates the song structure using its generators with respect to the current state of the sequencer. After the macro-level generation, it passes the relevant information to all its *Performer*s for them to generate the actual sequences of musical notes, such as melodic patterns and chords. Each performer produces the next sequence for the relevant bar using their generators to be played by *Instrument*s. However, the performers initially use abstract structures of notes—only meta information—rather than producing the concrete notes, so that, the notes can be modified accordingly when necessary in order to offer an adaptive outcome before they get played. *Conductor* takes that generated note sequence in each beat and transforms the notes according to the musical parameters determined by the user interactively. After the adjustment, the meta information gets converted to actual notes and are written into the musical score by the performer. Finally, each instrument plays all the notes in its performer's score, i.e., generates the final output which is audible to the user.

3.2. Hierarchical Music Composition

Generators in the engine jointly compose the musical pieces in three levels of abstraction, as shown in Figure 2. Firstly, *MacroGenerator* generates the musical form at macro level, i.e., it creates the main song structure which is made of a sequence of sections. Whenever the sequencer arrives to a new section, *MesoGenerator* generates the harmonic progression, i.e., a sequence of bars for that section. Unlike the pipeline architecture, these two levels are managed solely by the ensemble, hence, the generations are unified for all the performers. Thus, every performer in the ensemble obeys the same song structure when composing their individual scores. That being said, when a new bar arrives, each *MicroGenerator* of each performer generates a meta note sequence for that bar. The meta information of a note consists of a relative pitch index, relative offset (from the beginning of the bar), duration and volume of that note.

ModeGenerator generates the musical scale for the current state of the engine to be used when filling the musical score by the actual notes. The scale can be thought of as the pitch quantisation of the notes to be played. It is an array of ordered indices that represents the distance from the key note, hence the scale, of the piece. In that sense, the relative pitch index in a meta note refers to the note in that specific index in the array of that scale.

For the generation phase in all levels, there are a number

[1] An audio pulse refers to the lowest level of quantisation for sequencing the elements in the audio thread. It could be seen as the smallest significant interval for an audio element to be placed.

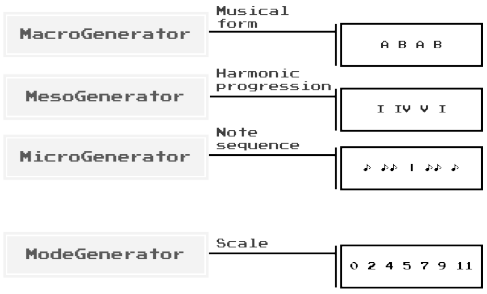

Fig. 2: Generators structural diagram (example iterations are given on the right).

of approaches and algorithms [25] which could be used in the field such as stochastic systems, machine learning, cellular automata and state-based architectures. The idea, is that our architecture lets the user decide what to use to accomplish the desired task. Therefore, all the generators are designed as abstract base classes to be implemented depending on the real-time choices of the user. This approach also promotes flexibility and encourages the developer-user to combine different techniques together, or use them interchangeably in the system even in run time. The first version of *barelyMusician* includes implementations of a number of different generators discussed above.

3.3. Rule-based Note Transformation

The *Conductor* component is used in order to achieve affective smooth transitions when required by manipulating the generated composition in real-time. The interactivity between the user and the interface is managed by *Musician* using two high-level parameters, namely *energy* and *stress*[2], which are meant to state *the current mood* of the system. The user is able to modify those parameters anytime to change the mood of the musical piece accordingly. The parameters are directly mapped to selected musical properties that are managed in the component, as shown in Figure 3. Therefore any change the user makes in the interface results in an immediate response in low-level musical parameters of the engine. This mechanism is achieved by using those parameters effectively during the note transformation phase. Since the transformation process is done in each sequenced beat, the outcome is capable of reacting even to a minor modification immediately in an adaptive manner.

The musical properties have been chosen with respect

[2]The terminology was adopted from the AUD.js [22] project.

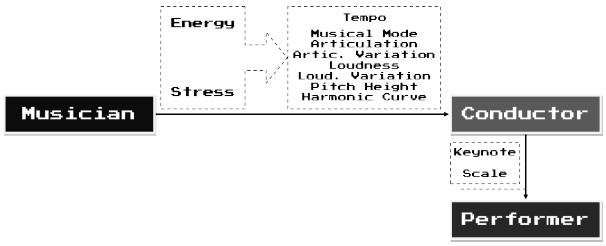

Fig. 3: Note transformation diagram.

to Livingstone et al.'s extensive research and analysis on emotion mapping [19]. While the selected parameters strictly follow their results, not all the findings have been used in this project in order to keep the design feasible enough for the implementation.

The mapping is done in two steps; first the normalized value of the relevant musical property is computed using the energy and stress values, then the value is scaled according to the specified interval of that property (see Table 1). For instance, to get the final tempo value for the playback, the tempo multiplier is calculated using the energy value. If the energy value is 1.0, then the tempo multiplier is computed by $0.85 + (1.0 * 1.0) * 0.3 = 1.15$. Finally, the resulting tempo multiplier is used to scale the initial tempo, say 120 BPM, resulting in $1.15 * 120 = 138$ BPM.

3.4. Audio Output Generation

Real-time sound synthesis and sampling techniques were chosen to be used to generate the audible output—rather than relying on more primitive technologies such as MIDI—in order to achieve a sufficiently acceptable quality of sound as would be seen in other practical applications in the industry. More specifically, the desired audio data is generated procedurally from scratch or loaded from a pre-existing sound bank—sample by sample—by the instruments in the composition phase.

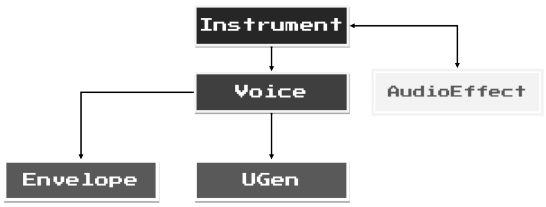

Fig. 4: Instrument components and dependencies.

Musical property	Interval	ΔEnergy	ΔStress
Tempo multiplier	[0.85, 1.15]	1.0	-
Musical mode	[0.0, 1.0]	-	1.0
Articulation multiplier	[0.25, 2.0]	-1.0	-
Articulation variance	[0.0, 0.15]	1.0	-
Loudness multiplier	[0.4, 1.0]	1.0	-
Loudness variance	[0.0, 0.25]	0.5	0.5
Pitch height	[-2, 1]	0.25	-0.75
Harmonic curve	[-1, 1]	-0.75	-0.25
Timbre brightness	[0.0, 1.0]	0.6	-0.4
Timbre tension	[0.0, 1.0]	0.8	0.2
Note onset	[0.25, 4.0]	0.5	0.5

Table 1: Mapping between the musical properties and the mood.

As shown in Figure 4, each instrument has a certain number of *voice*s that generate the actual output. Voice components are capable of creating the sound by their *unit generators* either using sound synthesis techniques or using pre-existing sound samples as their audio data. The sound is shaped by the *Envelope* component [26], when it gets played by the instrument. Furthermore, after the creation process, the output can further be manipulated using *AudioEffect* components. Added audio effects, such as filtering and reverberation, are applied to each voice in the instrument to produce the final output data.

3.5. Limitations

Certain simplifications had to be made due to the limitations (time and resource) and the broadness of the topic, in order to make the research project feasible. One major simplification that was made as a design choice is to restrict the music transformation process to micro level—for notes only. In other words, musical form is built regardless of the musical properties selected by the user during run time to prevent the potential issues that might occur by such complexity.

Moreover, it was decided to keep the key note of the composition intact throughout the piece without any key modulations. The main reason is the difficulty of preserving the consonance of a particular musical pattern, especially when the transformations take place. In fact, even when the scale remains the same, unintended dissonance is likely to occur in the piece when the key note changes. Thus, the harmonic progressions are limited to change only the musical mode of the piece without changing the key note of the scale. On the other hand, the key note could be changed manually anytime during play if intended by the user.

3.6. Implementation Features

The engine was implemented in C# making full use of the capabilities of the language in order to provide sufficient functionality in terms of the needs of the potential end-user. Unity3D game engine was chosen as the base framework considering not only its popularity but also portability and ease of use. An application programming interface (API) was developed, namely *BarelyAPI*, that enables a flexibility of the given features which can subsequently be integrated into a third party system. Moreover, a graphical user interface was included as part of the framework, so that, it allows an intuitive interaction for users with various technical backgrounds. We are currently preparing *barelyMusician* for release under an open source license. Having said that, the system could be used not only by developers but also audio designers or even music composers themselves.

Furthermore, additional components such as a keyboard controller and real-time audio recorder are provided inside the framework. One other supplementary feature worth mentioning is that the audio events in the engine are not only used internally but also could be accessed externally by using the programming interface. That being said, custom event listener functions can be easily registered to the sequencer to get relevant signals on each pulse, beat, bar and section respectively. A congruency between visual and audio elements in a virtual environment is very important for perception and emotional involvement. Even though the functionality is provided as an extra feature, its capabilities are powerful enough for the engine to be used as the main motivation.

4. EVALUATION

Two applications of the music system have been developed in order to test and evaluate the musical capabilities of the engine[3]. Firstly, a demo scene was developed with a graphical interface in order to test the interactive results of the engine. The second application involved the development of a simple game scenario in order to observe how the system—as is—could be used by a game developer. The development stage of these demonstrative applications was a crucial step in the project, as the whole process made it possible to spot the flaws and missing but desirable features clearly in terms of practical aspects before the evaluation phase. Examples of such late features include user input driven generators to provide a broader interaction and flexibility to the prospective end-user. Nevertheless, the current state of the engine is fully-functional and ready-to-use in any third party interactive system.

The primary focus of the evaluation was to assess the level of interactivity and recognition of mood that the user may experience when using the music generation system. Informal findings gathered from a preliminary qualitative user study based on the demo scene shows that real-time modifications done by the user result in clearly recognisable changes of the characteristics of the songs, i.e., over 95% of the 22 participants were immediately able to perceive the difference between settings of mood selections during playback. On the other hand, while such moods as *depressive* were spotted easily, some participants described others, such as *angry*, using the word *upset*. Therefore, it may be of interest to further study the abstraction of moods, especially the renaming and readjusting of values in order to clarify mood description. Certain emotions are known to be more difficult to be perceived by the listener [19], hence, further experimentation with different scenarios—e.g. with visuals and a narrative support—might be beneficial in the future to obtain more precise results. Nonetheless, the main focus of the approach does not take into account these specific abstractions, therefore, in terms of *energy* and *stress*, it can be concluded that the resulting work successfully generates interactive music with a smooth transition between recognisable emotional states.

In terms of the quality of the produced output, while the current state of the engine features a promising potential with its instrument architecture, more sophisticated audio synthesis and processing techniques should also be considered in order to achieve a musical outcome analogous to non-generative traditional musical pieces. However, it can be argued that the current music engine can produce acceptable compositions that in many ways are comparable to simple tonal Western music.

[3] A sample footage of the demo scene can be accessed through `http://youtu.be/Qs3yDVtqt9s`.

5. CONCLUSIONS & FUTURE WORK

A novel approach was proposed in this paper for combining real-time music generation and affective transformation methodologies into a single cohesive framework to be used in any type of interactive entertainment system in academia or industry. Moreover, the proposed music transformation method is unique in the field, as it allows manipulation of a musical piece in real-time at the sound-wave level.

We feel the work presented here constitutes an important step for potential practical uses of an interactive music system that responds to dynamic changes in higher level parameters. While the core engine is functionally complete, the resulting work should be considered only as an initial step. One major area of future work is to add more generators, instruments and audio effects to the engine. Particularly, more sophisticated generators with various capabilities would dramatically enhance the ease of use, so that the users would not necessarily have to have any musical background or programming skills while using the engine. Additionally, certain psychoacoustic parameters, such as for reverberation, are planned to be added to the system in order to further enrich the generated output—hence, the overall experience. Last but not least, augmenting the approach with spatialisation techniques remains a potentially interesting research subject to be studied in the future.

6. REFERENCES

[1] A. Robertson, "How 'the last of us' sound design outshines gameplay", July 2013 (accessed 10 October 2014), http://www.forbes.com/sites/andyrobertson/2013/07/04/the-last-of-us-in-depth-review-sound-design/.

[2] P. Vorderer and J. Bryant, *Playing Video Games: Motives, Responses, and Consequences*, LEA's communication series, Lawrence Erlbaum Associates, 2006.

[3] K. Larson, J. Hedges, and C. Mayer, "An adaptive, generative music system for games", *Game Developers Conference*, 2010.

[4] D. Valjalo, "Game music: The next generation", August 2013 (accessed 10 October, 2014). http://www.gamesindustry.biz/articles/2013-08-12-game-music-the-next-generation.

[5] N. Fournel, "Procedural audio for video games: Are we there yet?", *Game Developers Conference*, 2010.

[6] A. Harper, *Infinite Music: Imagining the Next Millennium of Human Music-Making*. John Hunt Publishing Limited, 2011.

[7] D. Tymoczko, *A Geometry of Music: Harmony and Counterpoint in the Extended Common Practice*. Oxford Studies in Music Theory, Oxford University Press, USA, 2011.

[8] F. Lerdahl, R. Jackendo, and R. Jackendo, *A Generative Theory of Tonal Music*, MIT Press series on cognitive theory and mental representation, MIT Press, 1983.

[9] L. Meyer, *Emotion and Meaning in Music*. Phoenix books, University of Chicago Press, 1956.

[10] G. Nierhaus, *Algorithmic Composition: Paradigms of Automated Music Generation*. Mathematics and Statistics, Springer, 2009.

[11] J. A. Maurer IV, "A brief history of algorithmic composition", 1999 (accessed on 10 October, 2014). http://ccrma.stanford.edu/ b̃lackrse/algorithm.html.

[12] M. Z. Land and P. N. McConnell, "Method and apparatus for dynamically composing music and sound eects using a computer entertainment system", May 24 1994. US Patent 5,315,057.

[13] Bush, Gershin, Klein, Boyd, and Shah, "The importance of audio in gaming: Investing in next generation sound", *Game Developers Conference*, 2007.

[14] A. Berndt, R. Dachselt, and R. Groh, "A survey of variation techniques for repetitive games music" in *Proceedings of the 7th Audio Mostly Conference: A Conference on Interaction with Sound*, pp. 61–67, ACM, 2012.

[15] M. Hoeberechts, R. J. Demopoulos, and M. Katchabaw, "A flexible music composition engine", *Audio Mostly*, 2007.

[16] M. Hoeberechts and J. Shantz, "Realtime emotional adaptation in automated composition" *Audio Mostly*, pp. 1–8, 2009.

[17] K. Jolly, "Usage of pure data in spore and darkspore" *in Pure Data Convention*, Weimar, 2011.

[18] E. Berlekamp, J. Conway, and R. Guy, *Winning Ways for Your Mathematical Plays*. No. v. 2, Taylor & Francis, 2003.

[19] S. R. Livingstone and A. R. Brown, "Dynamic response: Real-time adaptation for music emotion", in *Second Australasian conference on Interactive entertainment*, Sydney, pp. 105–111, Creativity & Cognition Studios Press, 2005.

[20] S. R. Livingstone, R. Muhlberger, A. R. Brown, and W. F. Thompson, "Changing musical emotion: A computational rule system for modifying score and performance", *Computer Music Journal*, vol. 34, no. 1, pp. 41–64, 2010.

[21] A. Friberg, "pdm: An expressive sequencer with real-time control of the kth music-performance rules", *Computer Music Journal*, vol. 30, pp. 37–48, Mar. 2006.

[22] F. K. Timothey Adam, Michael Haungs, "Procedurally generated, adaptive music for rapid game development", in *FDG 2014 Workshop Proceedings*, Foundation of Digital Games, 2014.

[23] M. Hoeberechts, R. Demopoulos, and M. Katchabaw, "Flexible music composition engine", Nov. 15 2011. US Patent 8,058,544.

[24] S. Bokesoy, "Presenting cosmosf as a case study of audio application design in openframeworks", in *International Computer Music Conference Proceedings*, vol. 2012, 2012.

[25] M. Edwards, "Algorithmic composition: computational thinking in music", *Communications of the ACM*, vol. 54, no. 7, pp. 58–67, 2011.

[26] R. Boulanger and V. Lazzarini, *The Audio Programming Book*. MIT Press, 2011.