Algorithmic music as intelligent game music

Conference or Workshop Item

Algorithmic Music As Intelligent Game Music

Anthony Prechtl and **Robin Laney** and **Alistair Willis** and **Robert Samuels**[1]

Abstract. Current game music systems typically involve the playback of prerecorded audio tracks which are crossfaded in response to game events such as level changes. However, crossfading can limit the expressive power of musical transitions, and can make fine grained structural variations difficult to achieve. We therefore describe an alternative approach in which music is algorithmically generated based on a set of high-level musical features that can be controlled in real-time according to a player's progression through a game narrative. We outline an implementation of the approach in an actual game, focusing primarily on how the music system traces the game's emotional narrative by periodically querying certain narrative parameters and adjusting the musical features of its output accordingly.

1 INTRODUCTION

Modern computer games feature increasingly dynamic and nonlinear narratives. In many cases there is an entire game world that evolves in different ways depending on player interaction and choices. Like in films, music in games is generally used to help set the mood and convey information about the narrative. However, in games, the exact course and timing of the narrative are not known in advance, so a fixed, prerecorded score is not particularly viable. To address this, most game music systems utilize a set of looped audio files which are crossfaded when the game state is changed [3]—for example, during a transition to a new level, or from a state of safety to one of danger—but it is not immediately clear how this can accurately and expressively trace the more minute intricacies of a narrative in the way that film scores can.

We will describe an alternative approach to game music whereby music is algorithmically generated from a set of variable, high-level musical features such as *tempo* and *tonal/atonal*. The features are then interpolated in response to measurable changes in a game's narrative. This provides game designers with the ability to musically express different narrative situations, including even subtle variations, as well as to make fluid musical transitions between them. At the same time, it allows players to interact with the music through the game narrative, with the music providing increased and finer grained feedback compared to the traditional crossfading approach.

After further examining the problem and surveying some related work, we will describe our approach by outlining an algorithmic music system we have implemented in a simple computer game. We will focus primarily on how the game's internal logic monitors the narrative and specifies desired musical features for the music generator, and how the music generator can respond to them in such a way that it seems the music is tailored to the player's real-time narrative progression.

[1] The Open University, UK, email: anthony.prechtl@open.ac.uk

2 BACKGROUND AND RELATED WORK

As we have suggested, most modern game soundtracks achieve nonlinearity by dynamically crossfading different audio files [3]. In this approach, one or more audio files are assigned to each possible state in the game narrative—however the game designer encodes it—and when the game player moves from one state to another, the first state's music fades out while the second state's music fades in. The main problem with this approach is that at any time during the transition, as Berndt points out, the neighbouring audio files "have to harmonise tonally and metrically with each other [...] in order to facilitate musically convincing cross-fades without unintended disharmony and rhythmic stumbling" [2, pp. 356]. Put simply, it is relatively easy for harmonies and rhythms to clash, so crossfading between two audio audio files that are "incompatible" in this regard can lead to dissonances which may not be implied by the narrative. To minimize this, there are several imperfect solutions: The crossfades could be made relatively quickly, but this could be distracting or even jarring for the game player, and would not strongly support smoother and more gradual narrative transitions. Alternatively, the overall musical variation could be restricted so that each file has compatible harmonies and rhythms, but this could limit musical and emotional expression.

Müller and Driedger [9] propose what is essentially an automatic DJ system as a solution to the crossfading problem. In their proposed system, there would be a database of audio files each tagged with a tempo as well as encodings of its emotional and harmonic content. When a transition to a new target emotional state is requested, the system would find the audio file whose tagged emotion is nearest to the target emotion, but with a similar tempo and similar harmonic content as those of the initial audio file. The system would then time stretch or compress the initial audio file so that its beat matches that of the target audio file, while simultaneously crossfading to the new file. Although this approach does address the problem of clashing harmonies and rhythms, its ability to do so effectively hinges on the harmonic and rhythmic similarity of the initial and target audio files, or else they would clash during transitions in the same way as they would while naively crossfading (albeit at least with their beats aligned). The main problem with this requirement is that, as we have previously suggested, restricting the range of harmonies and rhythms would probably limit not only the potential for strong musical variations, but also the emotional capacity of the music.

There are some techniques similar in scope to crossfading, but which involve the use of digital signal processing to morph the perceptual features of one audio file into those of another, rather than simply overlaying their amplitudes. Perhaps the most well-known approach is that of Serra [16], which he calls *spectral modeling synthesis* (SMS). In short, SMS works by decoding multiple audio signals into spectral frequencies frame-by-frame, then combining and recod-

ing them into one audio signal. A smooth morph is therefore achieved by starting with only the frequencies of the initial audio file, and then, between each frame, gradually interpolating towards the frequencies of the target audio file. However, SMS and similar audio morphing techniques aim to morph relatively low-level perceptual characteristics (in Serra's case, the frequency spectrum), rather than high-level, structural musical features, which, to our knowledge, has not been reported thus far.

3 GAME OVERVIEW

We have developed a simple game entitled *Escape Point* in order to implement and test our approach. *Escape Point* is a first-person game in which the player navigates and attempts to escape a 3D maze while avoiding enemies. If the player reaches the maze exit, the game is won; if any enemy collides with the player first, however, the game is lost.

Both the graphics and the gameplay were designed to be minimal, yet familiar to typical game players. Mazes—and more generally, the task of navigating through large, complex spaces—are, of course, common in computer games, as is the notion of enemies chasing the player. We preferred simplicity in game design not only to minimize development time, but also to make the game more accessible and free of complicated rules, as well as to provide greater experimental control when we later evaluate the approach empirically.

We wanted the music to express the emotional narrative of the game. We define this in terms of how close the player is to the enemies, since this corresponds roughly to the degree of danger the player faces of losing the game, as well as the degree of alarm a person might feel in a similar real-life situation. Specifically, we wanted the music to gradually become more or less intense as this variable changes. For example, if the player slowly moves closer to an enemy, pauses a certain distance away, and then quickly retreats, the music will mirror the level of danger the player is in: the intensity will slowly rise, continue briefly at the higher intensity, and then quickly fall. Crucially, this would reinforce the fluctuating degree of alarm the player would likely experience throughout the scenario, and additionally provide the player with more or less immediate feedback about the current state of the game.

4 MUSIC DESIGN

Escape Point was developed using the Unity game engine[2], which is widely used in both independent and commercial computer games. The game logic was coded in C#, as is common in Unity games, while the music engine was designed in Cycling 74's Max[3], with communication between Unity/C# and Max occurring via UDP. Additionally, the Max patch outputs MIDI data which is converted to audio using IK Multimedia's SampleTank[4], a popular software sampler. We have found this configuration—specifically the use of Max and SampleTank as opposed to working entirely in C#—optimal for rapid prototype development and testing; it is worth noting, however, that in commercial game development such a configuration would not be considered standard.

This configuration hints at a strong separation between the game logic and the music generator. We treat the music generator as an independent module that runs autonomously, but which, through the game's internal logic, a game designer could control any way he or she sees fit. In the following subsections, we will describe specifically how this process works in *Escape Point*, and also discuss how it can be generalized. A flowchart further illustrating the approach is shown in Figure 1.

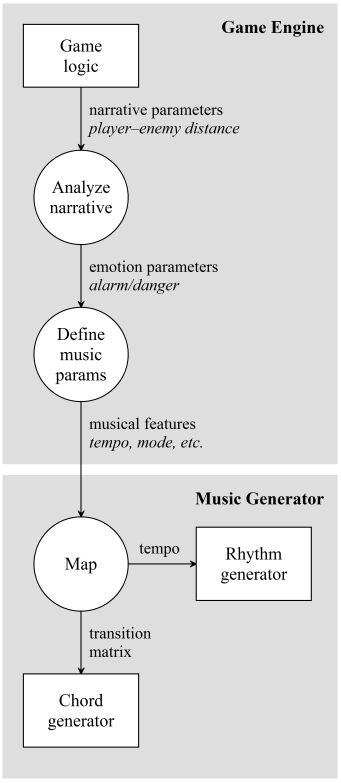

Figure 1. Flowchart showing how data progresses from the game engine to the music generator

4.1 Game logic responsibilities

The game's internal logic is responsible for controlling the music generator by specifying what its output should sound like. *Escape Point* does this by periodically analyzing the game narrative and sending a set of desired musical features to the music generator.

4.1.1 Narrative analysis

The first step in our approach is to devise a system that periodically queries the state of the game narrative, however the game designer chooses to codify it. In *Escape Point*, we are interested in the emotional narrative, which we consider to be the varying degree of danger or alarm that arises from proximity between the player and the enemies. We define the level of danger as a function of the distance between the player and the nearest enemy using the equation

$$danger = 1 - \frac{dist - dist_{min}}{dist_{max} - dist_{min}} \quad (1)$$

where $danger$ is a value between 0 and 1, and $dist$ is the distance between the player and the nearest enemy, which is clipped to a value between $dist_{min}$ and $dist_{max}$, the minimum and maximum distances to take into consideration, respectively. We use $dist_{min} = 3$

[2] http://www.unity3d.com
[3] http://www.cycling74.com/products/max/
[4] http://www.ikmultimedia.com/products/sampletank/

so that $danger = 1$ when the nearest enemy is within three metres of the player, and $dist_{max} = 25$ so that $danger = 0$ when the player is more than 25 metres from the nearest enemy.

Although we have found that just determining *danger* as described above is effective in capturing the relatively simplistic narrative of *Escape Point*, more advanced games would likely benefit from a more thorough narrative analysis. Strictly speaking, in *Escape Point*, the player is always in some danger, since there are no clearly denoted safe zones. This contrasts with games that distinguish between friendly and dangerous areas, for example, as well as games that have different levels with different emotional overtones.

4.1.2 Musical feature mappings

The second step in our approach is mapping the results of the narrative analysis to parameters for the music generation algorithm. We will discuss our algorithm in the following subsection, but Table 1 briefly outlines the musical features it accepts as parameters.

Table 1. Parameters used in the stochastic chord generation algorithm, all of which are continuous

Parameter	Controls
Tempo	The speed at which chords are generated
Velocity	The velocity with which chords are sounded
Volume	The overall volume of the synthesizer
Major chords	The probability of a major chord occurring
Minor chords	The probability of a minor chord occurring
Diminished chords	The probability of a diminished chord occurring
Dominant chords	The probability of a dominant chord occurring
Diatonic chords	The probability of a diatonic (within the keys of C major or A minor) chord occurring
Tonal chords	The probability of a chord occurring that has an obvious function in Western music theory (including non-diatonic chords) instead of one that does not
Functional transitions	The probability of a chord transition that is consistent with functional harmony instead of one that is not
Minor/major rules	Changes whether the above parameter favours the key of C major or A minor

In terms of the popular valence/arousal emotion space (see [14]), in which an emotion is encoded with a certain *valence* (how positive or negative the emotion is) and *arousal* (how aroused or calm the emotion is), we wanted the music to express increasingly negative valence and high arousal as the level of danger in the game increases. Fortunately, there is a large body of research concerned with correlating musical features with emotions that people perceive in the music. An overview is provided in [7] for primarily structural features such as mode and rhythm, and in [8] for primarily performance features such as articulation and loudness. Based on these, we decided that as *danger* increases, the following should occur:

- The tempo increases.
- The generated chords become louder.
- Major chords become less likely to occur.
- Diminished chords become more likely.
- Diatonic and other functional chords become less favoured.
- Functional chord transitions become less favoured.

Accordingly, a feature set is created with values for each of the parameters outlined in Table 1. The final step within the game logic is to send the set to the music generation algorithm, which, as we have already mentioned, occurs via a UDP connection between the game logic (C#) and Max.

4.2 Music generation

The core of the proposed approach is, of course, the music generator. It is responsible not only for being able to autonomously generate music, but also for being able to accept parameters in real-time and adjust its output accordingly. We have designed ours so that the parameters it accepts (shown in Table 1) are high-level musical features, which makes it relatively easy and intuitive to control, as was described in Section 4.1.2.

The generator uses a first-order Markov model to generate chords, as well as a constraint-based algorithm to voice lead them based on conventional rules in Western music theory. A *Markov model* is a stochastic model primarily comprised of a *transition matrix* whose elements describe the probabilities of each state in a system transitioning to each other state (see [1] for a full explanation). In our case, there are 48 states each representing a unique chord—for each of the twelve chromatic notes there are major, minor, diminished, and dominant seventh versions. Thus, the transition matrix is 48 rows by 48 columns, with each element $[i, j]$ defining the probability that the chord represented by i will transition to the chord represented by j. Whenever a new chord is requested, the system generates a pseudo-random number and uses it to choose the new chord based on the probabilities given by the row representing the currently sounding chord.

The *order* of a Markov model describes the number of previous states it takes into consideration for transitions to a new state. We use a *first-order* model—that is, the next chord is dependent only on the current chord—for several reasons: Compared to higher-order models, it requires a much smaller transition matrix, since the number of rows in the matrix increases exponentially as the order increases. Additionally, as Roads [13] points out, high-order Markov models "extend the window of local coherence over several events" (pp. 878). While this may sound desirable, it can lead to the system becoming "locked" into, and repeating, sequences of events, which might not be ideal when attempting to convey a dynamically changing narrative. On the other end, as opposed to a so-called *zeroth-order* model (i.e., not taking any chords into consideration when generating a new chord), a first-order model can encompass some of the conventions of Western chord progressions. For example, music theorist Walter Piston, in his "Table of Usual Root Progressions" [12, pp. 17], lists each of the seven diatonic scale degrees of a given key[5], describing roughly how often chords built on each one transition to chords built on other ones, which implies a first-order model. For example, he states that a I chord usually transitions to IV, sometimes to V, and less often to II or III. Similarly, composer Arnold Schoenberg [15] implies a first-order model by classifying intervals—in terms of the number of scale degrees—between neighbouring chords into three types of chord transitions: *strong* (or *ascending*), *descending*, and *superstrong*. Table 2 outlines these transition types and associated intervals.

Table 2. Schoenberg's [15] three types of chord transitions

Transition type	Interval
Strong (or ascending)	Four scale degrees up or three down
Descending	Four scale degrees down or three up
Superstrong	One scale degree up or down

[5] For example, in the key of C major, the scale degrees are C, D, E, F, G, A, B, each labeled from I–VII, respectively.

Markov models have been used extensively in music computing for a variety of tasks, including composition/generation (e.g., [1], [17], [5]), style imitation (e.g., [6], [4]), and interaction with a live performer (e.g., [11]). They are commonly created by analyzing a corpus or stream of music, counting the number of occurrences of each transition, and then constructing a transition matrix of probabilities or weights at which each transition occurs. One difficulty with using this approach for dynamic game music is that it is not immediately clear how to alter a trained Markov model so as to reflect a specific set of musical features, since it would already exhibit certain features of its own. A potential solution is to train multiple Markov models from different corpora, each with a unique set of musical features, and either interpolate the models or simply use the one whose features are closest to the desired musical features. However, we have found it simpler to instead generate Markov models directly (i.e., from scratch) from a given set of desired musical features. We do this by starting with a transition matrix in which all elements have an equal weight (in other words, the matrix implies entirely random transitions), and then filtering it based on the desired musical features. For example, if the *major chords* parameter is set to 0.25, then the weights of all transitions to major chords are multiplied by 0.25 so that—everything else being equal—there is relatively little chance of the system generating a major chord instead of a minor, diminished, or dominant one.[6] This process occurs consecutively for each parameter that directly affects the transition matrix (*tempo*, *velocity*, and *volume* do not).

The transition matrix of the Markov model varies over time depending on the parameters passed to the music generator. That is, whenever a new set of musical features is requested, the system discards the old transition matrix, creates a new one using the approach described above, and uses that one going forward. Of course, if the game narrative is idle, the system will simply remain in a steady state, outputting chords with the same probabilities. Otherwise, however, the probabilities will continuously change, always reflecting the current state of the game narrative. Similarly, the parameters of the music generator that do not directly affect the transition matrix—*tempo*, *velocity*, and *volume*—all vary in real-time.

Although we use a relatively simple Markov model, there are many other ways to generate music algorithmically (an extensive overview is provided in [10]). In theory, any algorithm with parameters that can be varied over time could be suitable. However, one major benefit of Markov models is that they are relatively intuitive in the sense that it is clear from a compositional standpoint exactly how their parameters—the probabilities that comprise the transition matrix—control the output, as opposed to, say, the parameters of a neural network. In practice, an algorithmic game music system would likely benefit from being both intuitive to control and sufficiently flexible so as to be able to convey a wide variety of emotional states, as well as to smoothly move between them.

5 CONCLUSION AND FUTURE WORK

We have shown how an algorithmic music system can respond in real-time to input from a game narrative: The state of the narrative is periodically analyzed and encoded, then mapped to musical features that are expressive of that state. The musical features are then sent to the music system, which updates its output accordingly. We have described a specific implementation of this approach that allows for both immediate and fine grained feedback, as well as for controlled transitions, all of which are difficult, if not impossible, in the currently conventional approach of crossfading audio files.

Our future work includes empirically evaluating the effectiveness of the proposed approach in comparison to more traditional approaches to game music. Perhaps most importantly, what remains to be seen is how it affects player enjoyment and perception of the game. Additionally, we wish to further develop our music generator, incorporating control over both melodies and rhythms elements instead of just harmonies, so as to allow for a greater range of musical and emotional expression.

[6] It is worth noting the slight distinction here between *probabilities* and *weights*: probabilities by definition must sum to 1, whereas weights need not. When filtering the transition matrix we treat the elements as weights, and afterwards convert to them to probabilities by ensuring all rows in the transition matrix sum to 1.

ACKNOWLEDGEMENTS

We would like to thank Dr. Andrew Milne at The MARCS Institute for helpful and insightful discussions.

REFERENCES

[1] Charles Ames, 'The Markov Process as a Compositional Model: A Survey and Tutorial', *Leonardo*, **22**(2), 175–187, (1989).
[2] Axel Berndt, 'Musical Nonlinearity in Interactive Narrative Environments', in *Proceedings of the International Computer Music Conference (ICMC 2009)*, eds., G Scavone, V Verfaille, and A da Silva, pp. 355–358, Montreal, Canada, (2009).
[3] Karen Collins, *Game Sound*, The MIT Press, Cambridge, Massachusetts, 2008.
[4] Tom Collins, Robin Laney, Alistair Willis, and Paul H. Garthwaite, 'Chopin, mazurkas and Markov', *Significance*, **8**(4), 154–159, (2011).
[5] Arne Eigenfeldt and Philippe Pasquier, 'Realtime Generation of Harmonic Progressions Using Controlled Markov Selection', in *Proceedings of the First International Conference on Computational Creativity*, Lisbon, Portugal, (2009).
[6] Mary Farbood and Bernd Schoner, 'Analysis and Synthesis of Palestrina-Style Counterpoint Using Markov Chains', in *Proceedings of the International Computer Music Conference*, Havana, Cuba, (2001).
[7] Alf Gabrielsson and Erik Lindström, 'The role of structure in the musical expression of emotions', in *Handbook of Music and Emotion: Theory, Research, Applications*, eds., Patrik N. Juslin and John A. Sloboda, 367–400, Oxford University Press, Oxford, England, (2010).
[8] Patrik N. Juslin and Renee Timmers, 'Expression and communication of emotion in music performance', in *Handbook of Music and Emotion: Theory, Research, Applications*, eds., Patrik N. Juslin and John A. Sloboda, 453–489, Oxford University Press, Oxford, England, (2010).
[9] Meinard Müller and Jonathan Driedger, 'Data-Driven Sound Track Generation', in *Multimodal Music Processing*, eds., Meinard Müller, Masataka Goto, and Markus Schedl, volume 3, 175–194, Dagstuhl Publishing, Saarbrücken/Wadern, Germany, (2012).
[10] Gerhard Nierhaus, *Algorithmic Music: Paradigms of Automatic Music Generation*, Springer-Verlag/Wien, New York, 2009.
[11] François Pachet, 'The Continuator: Musical Interaction With Style', in *Proceedings of the International Computer Music Conference*, Gothenburg, Sweden, (2002).
[12] Walter Piston, *Harmony*, Victor Gollancz Ltd, London, 1959.
[13] Curtis Roads, *The Computer Music Tutorial*, The MIT Press, Cambridge, Massachusetts, 1996.
[14] James A. Russell, 'A Circumplex Model of Affect', *Journal of Personality and Social Psychology*, **39**(6), 1161–1178, (1980).
[15] Arnold Schoenberg, *Structural Functions of Harmony*, Faber and Faber, London, second edn., 1983.
[16] Xavier Serra, *A system for sound analysis/transformation/synthesis based on a deterministic plus stochastic decomposition*, Ph.D. dissertation, Stanford University, 1989.
[17] Karsten Verbeurgt, Michael Dinolfo, and Mikhail Fayer, 'Extracting Patterns in Music for Composition via Markov Chains', in *IEA/AIE'2004: Proceedings of the 17th International Conference on Innovations in Applied Artificial Intelligence*, pp. 1123–1132. Springer-Verlag, (2004).