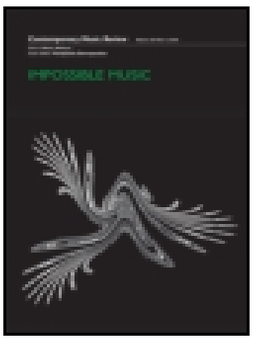

Routledge
Taylor & Francis Group

Contemporary Music Review

ISSN: 0749-4467 (Print) 1477-2256 (Online) Journal homepage: https://www.tandfonline.com/loi/gcmr20

An Introduction to Procedural Music in Video Games

Karen Collins

To cite this article: Karen Collins (2009) An Introduction to Procedural Music in Video Games, Contemporary Music Review, 28:1, 5-15, DOI: 10.1080/07494460802663983

To link to this article: https://doi.org/10.1080/07494460802663983

Contemporary Music Review
Vol. 28, No. 1, February 2009, pp. 5–15

Routledge
Taylor & Francis Group

An Introduction to Procedural Music in Video Games

Karen Collins

This article outlines some of the procedural music techniques that have been used, are being used, and may be used in the future in video games. The author examines different approaches to procedural composition and the control logics that have been used in the past, using a distinction between transformational and generative algorithms, and discusses reasons for the adoption of these techniques in games. She then examines why procedural music has not been more widely used by game composers and sound designers, and explores potential future directions in the area.

Keywords: Games; Algorithmic; Generative; Procedural; Interactive; Adaptive; Dynamic

Video games are perhaps an ideal media form for procedural music; after all, many elements of gameplay—especially the timing of events and actions—are unpredictable and occur in a non-linear fashion.[1] In a sense, it could be argued that all game soundtracks are procedural, in that the sonic elements that make up a game's overall audio soundscape (consisting of music, ambience, dialogue, sound effects and interface sounds—what I refer to collectively as *audio*) evolve in real time according to a series of rules set out in the game's software engine. Simple if/then algorithmic statements are behind the playback of most sonic events in games, such as: 'IF player picks up ITEM, THEN play ITEM jingle'. And of course, in games, even fixed linear recorded music is never played back twice in the same way, in the same order, with the same combination of other sound events. For instance, a player in a fight sequence may require a few extra sword swings on a second playthrough, or the player's character may die, or may encounter other in-game events at different timings, causing the music to cross-fade or segue into new cues, as well as triggering new or repeated sound effects and interface sounds, thereby changing the overall soundtrack.

The non-linear, variable elements in the sonic aspects of gameplay (what I have elsewhere defined as *dynamic* audio; see Collins, 2007) can be loosely divided into interactive and adaptive audio. *Interactive* audio refers to sound events *directly*

ISSN 0749-4467 (print)/ISSN 1477-2256 (online)
DOI: 10.1080/07494460802663983

triggered by the player, affected by the player's input device (controller, joystick, and so on). The player's input actions nearly always affect the soundscape of the game, from adding simple footsteps when the player clicks or holds movement buttons, to shooting sounds upon pressing a fire button, to triggering cross-fades based on the player's directional choices. In this way, the game offers a sound palette, and the player instigates changes in the overall soundscape's composition. For instance, if a player runs towards an enemy in a game while firing a weapon, the shots, the footsteps, and other sound effects may all be triggered by the player. The player may also encounter a 'trigger point' in the visual space of the game, whereby they change the music cue from one of relative peace to an 'enemy cue'. The player can typically run back and forth across the trigger point, re-triggering the enemy cue, until the enemy has been dealt with. The player, therefore, has control over these sound events in the game in terms of the timings of their playback, although the rules for playback (which sounds, associated with which events) are controlled by the game engine.

Adaptive audio events, on the other hand, are unaffected by the player's direct actions, although they are inevitably affected by indirect actions. Adaptive audio events are cued by the game's engine based on in-game parameters, and are therefore rarely immediately repeatable. These may include the general locations (as in place, such as 'in the village', or time of day, such as 'night', based on system clocks), scripted or unscripted events (it may be that the player needs to gather certain objects before a particular character will react in a specific way, for example), difficulty level, timing, player properties (skills, health, endurance), 'camera' angle, and so on. An example of adaptive audio is a change in music, ambience and sound effects as the timer in a game engine switches the scene from daytime to night-time based on the system clock. In the Nintendo game *The Legend of Zelda: Phantom Compass* (2006), for instance, there are pieces of mail that only arrive after a day has passed on the system's date calendar. While the player may exit the game and change the DS calendar and then reload the game to force these changes, these changes are intended as indirect (adaptive) actions. Like interactive audio events, these adaptive audio sound events are also generated in run-time by the game's scripts, but the actual timing of these events can be set by timers to execute at variable timings. Moreover, both interactive and adaptive sounds may be selected from a choice of many similar sounds in a database of stored sound effects, meaning that even if the player attempted to repeat the exact timing and playback of events, they may never be able to hear the same sounds in the same order twice.

With unpredictable event timings and player interactivity, game composition must be able to adapt quickly to the player's input (interactive audio) and to the computer-generated run-time game parameters (adaptive audio). In addition, games today tend to last far longer than linear (fixed) media forms (with often forty to sixty hours of scripted scenarios), and in the case of online multiplayer games where players interact with each other, the games are even more unscripted and open-ended. Many players engage in long gameplay sessions in which listening can become tiring, especially if a player spends a long time on one particularly difficult area of the game. In these cases,

players may turn off the sound altogether, or may substitute in their own music playlist (a feature on all Xbox360 titles). Some games have now incorporated timings into the music cues, so that if a player does get stuck on a level, the music will not loop endlessly, but will instead fade out. Or, as in the case of Electronic Arts' *Spore* (2008), the density of the instrumentation in the procedural music is reduced over time. Procedural music soundtracks therefore may offer some interesting possibilities that may solve some of these complications of composing for games. On the other hand, procedural music composers are faced with a particular difficulty when creating for video games: the sound in a game must accompany an image as part of a narrative, meaning sound must fulfill particular functions in games. These functions include anticipating action, drawing attention, serving as leitmotif, creating emotion, representing a sense of time and place, signaling reward, and so on (see Collins, 2008, pp. 123–138). Cues need to relate to each other, to the gameplay level, to the narrative, to the game's run-time parameters, and even to other games in the case of episodic games or those that are part of a larger series. Procedural music and sound in (most) games, therefore, must be bound by quite strict control logics (the commands or rules that control playback) in order to function adequately.

This article will outline some of the procedural audio techniques that have been used, are being used, and may be used in the future in video games. I will examine a series of different approaches to procedural composition and the control logics that have been used. I will then examine why procedural audio has not been more widely adopted by game composers and sound designers.[2] First, however, it is worth briefly mentioning the procedural and generative audio processes in games that are outside the scope of this article, including games as composition devices or instruments, online jam sessions, and procedural/generative music-based games.

Japanese composer and artist Toshio Iwai has been behind some of the most innovative procedural techniques in game audio. As early as 1987's *Otocky* (ASCII Corporation) for the Famicom (a Japanese Nintendo product similar to the Nintendo Entertainment System [NES]), Iwai incorporated procedural techniques into game design. In *Otocky*, a side-scrolling shooter, the player controls a ship that fires round balls at enemy shapes and flying musical notes. With a simple two-note bass line, the player's firing actions and in-game mechanics become the melody part, quantized in real-time to the beat.[3] Iwai later (1996) created *SimTunes* (Maxis), in which players paint onto the screen with different colours, where each colour represents a note. Insects (which each represent an instrument sound) then crawl over the colours and play the music. Arguably a composition device rather than a game in the traditional sense, *SimTunes* was an important precursor to later music-based 'games', such as *Electroplankton* for the Nintendo DS (2005), in which tiny plankton that generate tones are controlled by the player in various ways. With no set objectives, rewards or in-game narratives, these and similar interactive tools—while sold and marketed in the games industry—are arguably not games, but rather musical toys that use game interfaces (see Herber, 2008; Magnus, 2007). There are also musical composition devices and instruments that use game interfaces, ranging from MIDI controllers

created from game hardware to synthesizers and tracker sequencers built using the game hardware as interface device. These have often been designed for the production of 'chiptunes' music which uses (typically 8-bit) game hardware and/or software emulators with simple waveforms to produce music, but have sometimes incorporated procedural music techniques. Examples are *Nanoloop* and *Little Sound DJ* (both GameBoy music synthesizers/sequencers) and *Midines*, a Nintendo Entertainment System MIDI sequencer. Since neither musical toys nor composition interfaces are video games in the traditional sense, these are outside the scope of this article.

Similar to using game hardware or software to compose music, recently the Internet has allowed for spontaneous online jam sessions within the context of massively multi-player online games with varying degrees of control over the music between player avatars, such as in the Turbine game *The Lord of the Rings Online: Shadows of Angmar* (2007). The music, which is usually pre-composed, may be influenced by the players' selection of instruments, or players may pick up virtual instruments and play however they may wish. Similar jam sessions exist in other non-gaming online communities, in which the music may be entirely user-generated, such as in *Second Life* (see England & Wolek, 2007). There are also some very interesting possibilities hinted at in some recent games whose gameplay levels or imagery are generated by the player's choice of input music, such as the Harmonix game *Phase* (2008), which takes a user-generated playlist from an iPhone and creates game maps in a *Guitar-Hero* style (Harmonix 2005), or *Vib Ribbon* (Sony 1999), a game where a player's character runs along paths generated by input music.[4] In these cases, it is not the music that is being generated, but rather the game maps, and as such these are also outside the scope of this article.

A Survey of Procedural Music in Games

Before discussing a historical overview of procedural music in games, it is worth drawing on Wooller et al. (2005) to make a distinction between the degrees of procedural process in game music. Wooller et al. distinguish between what they call *transformational* algorithms and *generative* algorithms. Transformational algorithms have a lesser impact on the data size, but impact the overall structure. For example, a phrase may have several notes whose pitch value can be randomly altered, or phrases themselves may be restructured in the wider song, while the actual phrase remains unaltered. Within a phrase, instrument parts may be added or dropped according to in-game states. Generative algorithms, on the other hand, increase the overall musical data size in that the basic musical materials are themselves created. Due to the difficulties in composing effective procedural music for games, the vast majority of algorithms controlling music in games discussed here are transformational, rather than truly generative in the traditional sense (and in the terms of Wooller et al.).

There are several transformational possibilities used in game music, including the addition or subtraction of layers of instrumentation or tempo changes (such as an

increase in tempo in *Super Mario Bros*. as time runs out on the player). One transformational algorithmic process that has become increasingly common in games is the use of open form, or *recombinatorial* music. In open form, a composition's order of sequences or sections of music is left to the performer or chance, such as in Mozart's musical dice game, Stockhausen's *Klavierstück XI* (1956) or Henry Cowell's *String Quartet no. 3* (1935). The difference in the use of the technique in games, however, is that there is no 'performer' to make a conscious decision as to the order of playback, and the aleatoric element has been pre-programmed into the game engine so that an algorithmically controlled recombinatorial playback of cues occurs. In this sense, the composition's *structure* is generated by the computer, or in some fashion dependent on a player's input, and therefore the order and the timings of the cues are variable, but the elements within the cues (the sequences or phrases) are not themselves algorithmically generated. While it is possible for a recombinatorial piece to be viewed as generative, the size of the sequences used in games thus far has meant that the procedural element only applies structural transformation.

The Legend of Zelda: Ocarina of Time, for instance, used an open-form structure in the Hyrule field section of the gameplay. The game's map could be said to loosely resemble a wheel, with Hyrule field representing the hub. Since the player must spend a considerable amount of time running across Hyrule field to gain access to other important gameplay areas, and a standard repeated loop would become too repetitive, a series of cues are selected based on a random-number generator to maintain interest and diversity. Every time the game is played, the song played during the Hyrule field parts of gameplay plays back differently. This technique had also been used in much earlier games, when constraints of memory and cartridge/disk space meant that musical looping became common in games. A few composers at the time attempted to randomize different elements of music play-back to combat the repetitive nature of loops. The 1988 game *Times of Lore* (Martin Galway, Microprose) for the Commodore 64 used a selection of guitar solos whose sequence order was selected based on random-number generators. In this way, the game's ten songs (over thirty minutes of music) could fit into just 923 bytes.

A related form of transformational algorithms used in game composition has been the smaller parameter-based changes that have been programmed into larger sequences of music, defining start and end points, jumps, repeats/loops, and so on. In these cases, a composer creates musical sequences, and then conditionalizes those sequences, making decisions involving which segments should play continuously, change, be enabled or disabled, loop, converge, swap instruments, branch, jump, and so on. Markers, or decision points, indicate places where changes or branches in the performance may occur based on a condition in the game. For example, the patent for iMuse, a music engine designed at LucasArts in the early 1990s, describes the process as follows:

> … fight music, rather than playing along unresponsively, can be made to change [the] mood of the game in response to the specific events of the fight. For example,

> certain instrument parts can signify doing well (for example, a trumpet fanfare when a punch has been successfully landed), while others can signify doing poorly (for example, staccato strings when the fighter has taken a punch) . . .Also, it may be desirable to transpose . . . the music as the fight reaches its climax. This can also be done either immediately under entertainment system control, or by hook message (if it is necessary that the transposition occur only at an appropriate point in the music sequence). The resulting fight music will change mood and character along with the intensity and excitement of the fight, but in a smooth and aesthetically natural way, much like the music would follow the action in a feature length motion picture. (Land & McConnell, 1994, p. 5)

Although the game engine cannot necessarily always predict what actions a player may take, the illusion can be created if music can respond to various parameters. In the above case, for instance, the music could be altered according to player and enemy's health meters, such as a simple 'IF player's health is more than enemy health, THEN play WINNING music. IF player's health is less than enemy health, THEN play LOSING music.' The music can be controlled by a jump cue—'jump to winning section of song'—or could be altered on the basis of instrumentation changes—'add trumpet fanfare'.

The logical extension to parameterized playback of such larger sequences of music is transition matrices with complex control logics that manage small segments of music. Depending on a game's parameters, smaller segments can be called upon to serve as transitional segments or to create new musical elements within a piece. Such an approach to composition can be very effective in a game, although the planning required means constraints on time that may be prohibitive. Guy Whitmore used a transition matrix in the Monolith game *No One Lives Forever* (2000), which had several hundred small musical segments that made up six different emotional states for every given part of the game. Depending on various in-game parameters (such as the number of enemies present), the music could ramp up to intense combat music or drop down to ambient levels.

In addition to these transformational algorithmic-controlled examples of game music, there have been several projects to incorporate what Wooller et al. (2005) define as algorithmic generative music. Lucasfilm Games' *Ballblazer* (1984), for instance, employed what composer Peter Langston termed the 'riffology' algorithm, in which dynamically weighted choices for parameters are decided by the computer:

> such as which riff from a repertoire of 32 eight-note melody fragments to play next, how fast to play it, how loud to play it, when to omit or elide notes, when to insert a rhythmic break, and other such choices. To choose the next riff to play, the program selects a few possibilities randomly (the ones that 'come to mind' in the model). From these it selects the riff that is 'easiest' to play, i.e. the riff whose starting note is closest to one scale step away from the previous riff's ending note. To decide whether to skip a note in a riff (by replacing it with a rest or lengthening the previous note's duration) a dynamic probability is generated. That probability starts at a low value, rises to a peak near the middle of the solo, and drops back to a low value at the end. The effect is that solos start with a blur of notes, get a little lazy

> toward the middle and then pick up energy again for the ending. The solo is accompanied by a bass line, rhythm pattern, and chords which vary less randomly but with similar choices. The result is an infinite, non-repeating improvisation over a non-repeating, but soon familiar, accompaniment. (Langston, 1986)

Langston claims that while the song passes the 'is it music' test, it fails to pass the 'is it interesting music' test, 'because the rhythmic structure and the large scale melodic structure are boring. It appears that for music to remain *interesting* it must have appropriate structure on many levels' (Langston, 1986).

Due to the fact that most games have a set narrative with strategic moods and plot points, it has been rare that games have used these more advanced types of procedural music. There have been a few cases, however, where, due to the more abstract nature of gameplay in these games, procedural music has been an interesting choice. In the artificial life *Creatures* series (Millennium Interactive), for instance, starting with the third game in the series (music by Peter Chilvers 1997), music was recorded in short samples of as little as a note or a chord, and scripts controlled the logic, mapped to in-game run-time parameters (actions, environments, mood, and so on). Each creature in the game had its own set of instrument samples which would influence the music in a given area based on interactions with each other and the location, whose music was typically generated by feedback loops. Not tying actions directly to musical events meant that the music could have a more organic feel (see Creatures Labs, 2000; Parry, 2004).

A similar approach is taken in *Spore*, designed by Will Wright (*The Sims*), with music by Brian Eno (programmed with Kent Jolly and Aaron McLeran). *Spore* is comparable to *Creatures* in many ways, in that the entire game relies heavily on procedural generation to grow creatures from a cellular level to the creation of entire worlds (see also Herber, 2008). After being prototyped in Max/MSP, the music was created in Pure Data and consists of many tiny samples which generate the soundtrack in real time. Melodies and rhythms are all generated within limited rules; for example, a sequence might use notes from only within one specific scale. The player can also construct and edit their own music, including various anthems for cities, by selecting from a number of rhythmic sequences and note samples. Mechanisms were put into place to limit the player's input, however, so that changes would not happen too radically and the pieces could 'make sense' musically.[5]

New procedural possibilities have also been opened up by online multiplayer games, in which the number of players (each representing an instrument sound) in any given scene can be altered by the number of players online. In these cases, it has been possible for players to influence the music by their mere presence. In some cases, the interaction between players alters the key or mode in which the music is played, as in Turbine's *Asheron's Call 2: Fallen Kings* (2002). In this game, changes in mode indicated changes in gameplay and gave the player a sense of awareness of their predicament at that point in time: 'A player standing alone in an area would hear the background score in the Lydian mode, but as creatures fill the area, the music transitioned to Phrygian, giving a darker tonality. If monsters continue to outnumber

the player, the music might transition to a diminished mode while a quickening drum groove begins to pulse' (Booth, 2004, p. 480).

Where Procedural Music in Games is Going, and Why it Hasn't Got There Yet

Procedural music is clearly advantageous to some games, and there are tools available to facilitate its use, so why hasn't it been more readily adopted? The most obvious reason already mentioned is that procedural music in a game must be *meaningful* in some way. But while the functions of music in a game—particularly the important role of immersing the player in the game through emotional induction—is probably the most important reason why procedural music has not been more widely adopted in games, there are also other reasons. One of the largest fears since the advent of Redbook audio in games in the late 1980s is that a return to MIDI will disappoint audiences who have now grown accustomed to high-quality samples and even live orchestras in games. Although MIDI has advanced considerably in its adoption of the downloadable sound (DLS) specification, there is still a common perception that MIDI means poor-quality music. Another important reason for the lack of adoption is that procedural audio tends to be expensive CPU-wise, in a world in which audio is rarely given priority against graphics or other elements that must compete for the same resources. Similarly, games often do not have budgets or scheduling to allow for the time that it takes to instigate complex control logics (which result in higher engineering bills). Another issue slowing the adoption of these techniques is the music education system, which favours linear music production. Many game composers were schooled in linear media composition such as film music, rather than the more dynamic styles which may be offered as part of a contemporary composer's training, but are rarely introduced at undergraduate level.[6] Moreover, there are many difficulties in communicating what procedural content *is* (and its value) to the player (and, often, to the game's producers): in other words, marketing the advantages of procedural audio is difficult to an audience that may not understand what it means (Humphrey, 2008, p. 5).

Despite these difficulties, the increasing adoption of procedural creation of in-game graphics and physics hints at a coming world for more procedural audio content in games as well. The procedural generation of gameplay levels has been used at least as early as *The Sentinel* (music by Geoff Crammond 1986). The 2004 German game *.kkrieger* (.theprodukkt) was entirely procedurally generated, creating a first-person shooter in just 97 kB. A recent article on procedural generation in casual games lists 'more content for less money' and 'more replayability' as key reasons for adopting procedural content (that is, graphics, level maps or music; see Humphrey, 2008, p. 5).

There has also quite recently been an increase in procedural audio systems designed for multimedia, particularly film, although these currently experience very little adoption. The *Mkmusic* system (Mishra, 1999), for instance, was designed for generation of music for animation. The *QSketcher* and *EMuse* programs similarly are designed to accompany video or film (Abrams et al., 2002; Vane, 2006; see also

Hedemann et al., 2008; Jewell et al., 2003), and Sony's recent *Cinescore* program likewise provides users with the ability to adjust audio clips set to film. While these programs have been designed for linear media forms, the fact that Sony is now producing such music software indicates an increasing awareness of and desire for these types of programs. There have also been a few early attempts at software for procedural audio in games (such as Veneri & Planqueel's *Play All* [Veneri & Planqueel, 2008]), but more often existing programs such as Pure Data are being bridged to game engines, as was the case with *Spore*.

It is likely that we will soon see more procedural music in games, particularly as mood cues tagged to certain in-game events that may alter pre-defined themes or gestures as an elaboration of the original phrase or musical sequence.[7] In such cases, the entire clip does not have to be procedurally generated, but rather, algorithms could control various instruments, melody lines, or sections of the piece. For instance, there could be a drum and bass rhythm which is improvised upon (the riffology method), a change in drum pattern which is randomly developed as time progresses, or a basic melody which is harmonically orchestrated by the computer based on mood tagging. Indeed, although the procedural methods that have been used in games thus far have been rarely implemented, as work continues to advance in this area, there are many procedural methods which look to be quite promising in their potential application in games in the future.

Notes

[1] I define procedural music as composition that evolves in real time according to a specific set of rules or control logics. As shown, this can take the form of generative composition or transformational composition, the line between which can be somewhat indistinct.

[2] While sound effects can be generated or procedurally altered (synthesized or altered in real time) in some way using a variety of synthesis methods, these techniques are outside the scope of this article. See Paul (2008) for an introduction to granular synthesis as it may be applied specifically in games.

[3] A similar use of quantized notes created by player-controlled projectiles was used in the game *Rez* (created by Tetsuya Mixuguchi in 2001/2002 on the PlayStation 2).

[4] There are now software engines that combine music information retrieval techniques to incorporate user-generated playlists into games. See, for instance, the Echo Nest Analyze API (http://the.echonest.com/analyze.html).

[5] Aaron McLeran, 'Procedural Music in Spore' presentation at the Game Developer's Conference, San Francisco, February 2008.

[6] For a breakdown of obstructions to the implementation of procedural sound effects, see Farnell, 2007.

[7] There has been considerable work done in the area of affect, semiotics and mood-tagging. See, for instance, Birchfield, 2003; Eladhari et al., 2006; Livingstone & Brown, 2005.

References

Abrams, S., Bellofatto, R., Fuhrer, R., Oppenheim, D., Wright, J., Boulanger, R., Leonard, N., Mash, D., Rendish, M. & Smith, J. (2002). QSketcher: An environment for composing music for film. In *Proceedings of Creativity & Cognition* (pp. 157–164). New York: ACM Press.

Birchfield, D. (2003). Generative model for the creation of musical emotion, meaning, and form. In *ACM SIGMM Workshop on Experiential Telepresence* (pp. 99–104). New York: ACM Press.

Booth, J. (2004). A DirectMusic case study for *Asheron's Call 2: The Fallen Kings*. In T. M. Fay, S. Selfon & T. J. Fay (Eds.), *DirectX 9 audio exposed: Interactive audio development* (pp. 473–498). Plano, TX: Wordware Publishing.

Collins, K. (2007). An introduction to the participatory and non-linear aspects of video games audio. In S. Hawkins & J. Richardson (Eds.), *Essays on sound and vision* (pp. 263–298). Helsinki: Helsinki University Press.

Collins, K. (2008). *Game sound: An introduction to the history, theory, and practice of video game music and sound design.* Cambridge, MA: MIT Press.

Creatures Labs. (2000). The music behind *Creatures.* http://www.gamewaredevelopment.co.uk/creatures_more.php?id=459_0_6_0_M27.

Eladhari, M., Nieuwdorp, R. & Fridenfalk, M. (2006). The soundtrack of your mind: Mind music—adaptive audio for game characters. In *Proceedings of the 2006 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology.* New York: ACM.

England, M. & Wolek, N. (2007). The online graphical community as a potential interface for interactive music performance. http://miditron.blogspot.com/ and http://sound.oneemptyspace.net/links.html.

Farnell, A. (2007). An introduction to procedural audio and its application in computer games. http://obiwannabe.co.uk/html/papers/proc-audio/proc-audio.html.

Hedemann, C., Sorensen, A. & Brown, A. R. (2008). Metascore: User interface design for generative film scoring. In *Proceedings of Sound : Space—The Australasian Computer Music Conference* (pp. 25–30). Sydney. Australasian Computer Music Association (ACMA).

Herber, N. (2008). The composition-instrument: Emergence, improvisation and interaction in games and new media. In K. Collins (Ed.), *From Pac-Man to pop music: Interactive audio in games and new media* (pp. 103–125). Aldershot: Ashgate.

Humphrey, A. (2008). Algorithmic content in casual games. *Casual Games Quarterly*, 3(1). http://www.igda.org/casual.

Jewell, M. O., Nixon, M. & Prugel-Bennett, A. (2003). CBS: A concept-based sequencer for soundtrack composition. In *Proceedings of 3rd International Conference on Web Delivering of Music* (pp. 105–108). IEEE Computer Society.

Land, M. Z. & McConnell, P. N. (1994). Method and apparatus for dynamically composing music and sound effect using a computer entertainment system. US Patent No. 5,315,057.

Langston P. (1986). (201) 644-2332 or Eedie & Eddie on the wire: An experiment in music generation. http://www.langston.com/Papers/2332.pdf.

Livingstone, S. R. & Brown, A. R. (2005). Dynamic response: Real-time adaptation for music emotion. In *Proceedings of the Australasian Conference on Interactive Entertainment* (pp. 105–111). New York: ACM.

Magnus, C. (2007). Aesthetics, score generation, and sonification in a game piece. In *Proceedings of the International Computer Music Conference* (pp. 262–265). San Francisco: International Computer Music Association.

Mishra, S. (1999). *The 'mkmusic' system: Automated soundtrack generation for computer animations and virtual environments.* Unpublished MSc thesis, George Washington University, Washington, DC.

Parry, L. (2004). *A scripted sample-based music system for game environments using .NET.* Unpublished BSc thesis, University of Bath, UK.

Paul, L. (2008). An introduction to granular synthesis in video games. In K. Collins (Ed.), *From Pac-Man to pop music: Interactive audio in games and new media* (pp. 135–149). Aldershot: Ashgate.

Vane, R. E. (2006). *Composer-centered computer-aided soundtrack composition.* Unpublished MSc thesis, University of Waterloo, Canada.

Veneri, O., Planqueel, Y. (2008, June). *Create a scalable and creative audio environment: Middleware project play all.* Paper presented at the Paris Game Developer's Conference.

Wooller, R., Brown, A. R., Miranda, E., Berry, R. & Diederich, J. (2005). A framework for comparison of processes in algorithmic music systems. In *Generative arts practice* (pp. 109–124). Sydney: Creativity and Cognition Studios Press.