Interacting with Generative Music through Live Coding

Author

Brown, Andrew R, Sorensen, Andrew

Published

2009

Journal Title

Contemporary Music Review

DOI

10.1080/07494460802663991

Interacting with generative music through live coding

Andrew R. Brown and Andrew Sorensen
Queensland University of Technology &
Australasian CRC for Interaction Design

Keywords: live coding, performance, algorithm, interaction

Abstract
All music performances are generative to the extent that the actions of performers produce musical sounds, but in this article we focus on performative interaction with generative music in a more compositional sense. In particular we discuss how live coding of music involve the building and management of generative processes. We suggest that the human interaction with generative processes that occurs in live coding provides a unique perspective on the generative music landscape, especially significant is the way in which generative algorithms are represented in code to best afford interaction and modification during performance. We also discuss the features of generative processes that make them more or less suitable for live coding performances. We situate live coding practice within historical and theoretical contexts and ground the discussion with regular reference to our experiences performing in the live coding duo *aa-cell*.

Introduction

The practice of live coding involves writing and modifying computer programs that generate music in real-time. Often this music making activity occurs in a live performance situation with the code source projected for the audience. While it is possible to trigger sound events directly while live coding it is much more efficient to create generative processes that autonomously make music, freeing the performer to build or modify code for the next stage of the performance. Therefore, being able to efficiently describe generative processes as software is a primary concern for the live coder. In particular we have found that the following criteria apply to generative processes for use in live coding. Generative processes should be:

- succinct and quick to type
- widely applicable to a variety of musical circumstances
- computationally efficient allowing real-time evaluation
- responsive and adaptive by minimising future commitments
- modifiable through the exposure of appropriate parameters

For the live coder, software code is a medium of expression through which creative ideas are articulated and in this article we will focus on how generative processes facilitate that expression. Computer programming code is the music notation of live coding performances (McLean 2004). The code represents the musician's ideas and describes the musical outcome to the computer and to those with the knowledge to read it. Notated in code, the music is available for reflection, reuse and modification. It can be saved, replayed, shared with others or analysed. This affords reflection in action that has been shown to be highly valuable in creative tasks in many domains (Schön 1987) and generative processes facilitate this in ways not previously possible in live performance. In performance, the code is often modified substantially and therefore that which remains at the

end is often insufficient to represent the whole performance. For a full transcription of a live coding performance all of the changes would need to be logged. There are many other aspects of live coding performance beyond those discussed here and for further details of the history and scope of live coding practices the reader is directed to other literature (Collins et al. 2003, Brown 2006).

The authors have several years of experience in live coding, primarily performing as the duo *aa-cell* in numerous concerts. Originally our live coding practice was purely musical but in recent times our performances have included live coding of visual material as well. During performances the visual material and code are usually projected for the audience to view. Figure 1 shows *aa-cell* performing at a concert in Brisbane, Australia in 2008, and figure 2 shows a snap shot of the projected image during the performance.

Figure 1. The live coding duo aa-cell in performance.

Figure 2. The screen projected for audiences to view during an aa-cell concert.

Direct and meta creation

How is live coding a musical activity? Banging a drum creates sound, and potentially music, through a direct connection between a human action or gesture and a sound source. Only in a trivial sense can this be considered generative, and it is possible in live coding performances to emulate this direct creation by manually triggering the evaluation of one-shot code fragments. However, if live coding were limited to this direct evaluation it would be severely crippled as an expressive practice. In this article we are more concerned with generative music in an expansive sense, where substantial musical outputs are produced by an algorithm. Typically, there is considerable leverage in this process where the effort of describing the algorithm is minimal compared to the effort that would have been required (if it were even possible) to describe directly the musical material that is produced. Generative music practices have, therefore, a link with conceptual visual art and formalist music composition movements from the 20th century. These practices all share an emphasis on process as a creative driver and an indirect mediation between artist and artistic form, a process described as metacreation by Mitchell Whitelaw (2004).

In live coding concerts, performers often set up computational processes that generate an ongoing stream of musical output allowing the performer to turn their attention to writing or modifying processes. The computer plays a mediating role by executing the process and producing the sound. It is the ability to harness generative material that allows live coding performers to participate in a new kind of performance where they exercise indirect, or meta, control over the creation of their music.

Stylistically, much of the music produced by live coding has aesthetic links to other electronic music genres including musique concrète, electronic dance music, glitch music, and noise art; a grouping that is not overly coherent from the point of

view of aesthetics, but shares a concern for processes involving electronic sound systems. While performing with generative processes is relatively novel within this community, there seems to be an openness to it not found in many other music communities. This openness is likely a result of both the valuing of experimentation or novelty as well as the historical reliance on recorded sound for composition and performance, including DJing. Music making is understood as a construction and moulding of sound materials, rather than as a transcription of the musician's imagination. There is also an active interest in the role of technologies as drivers of innovation. Within this receptive environment it is not surprising that live coding activities have begun to flourish and spread.

Tools and technologies

Marcus Pearce and his colleagues (2002) articulate a sentiment about the role of coding in music making that resonates with us and many other computer musicians; that the development of software is a central component of computer music making.

> If a computer program is written by the composer, the development of the program is an integral part of the compositional process (since ultimately it is driven by the same motivations). Since the program is designed solely for use by its developer, there are no methodological constraints placed on its construction. Furthermore, there is no need to define any rigorous criteria for success nor to use such criteria in evaluating the program and the compositions. If the composer intends the music for public consumption, then they may only be evaluated in the same way that composers and compositions are usually appraised: through audience reactions at performances, record sales, critical reviews and so on. (Pearce et al. 2002 p.123.)

In live coding practice this view of software development as music making finds its logical, if perhaps crazy to some, conclusion: writing software becomes part of the performance. Code becomes the musical score; a score that is written, modified and executed as part of the performance. In this way live coding practices combine aspects of composition, arranging, improvisation and performance. The code acts as an intermediary between imagination and sound and is elaborated and transformed in performance within a tight feedback loop. This has similarities to the real-time interaction between musicians and prepared algorithmic processes, described as 'hyperimprovisation' by Roger Dean (2003). The use of generative processes in these practices is clearly significant for both the efficient production of output and in allowing the performer to reflect and respond. In support of this practice there are live coding tools that combine features of software development environments with digital sound synthesis and music data protocols. Almost any computer programming language can be used for generative music making and the affordances of computing languages for music have been explored in some detail elsewhere (Loy and Abbott 1985). For live coding purposes, languages are usually extended to include features for communicating with external synthesizers or for direct signal processing.

Dynamic programming environments are essential for live coding, they allow the interactive writing and evaluation of code segments on-the-fly. These environments have been around since at least the 1960s with systems using the LISP programming language but have become increasingly popular in recent years

as computing performance has mitigated their speed liability. As well as being dynamic, systems for musical live coding prioritise event scheduling to an extent not usually found in general purpose languages. This focus on accuracy of timing is obviously crucial for many media arts practices, particularly music. Popular environments for live coding music include SuperCollider (McCartney 1996), ChucK (Wang and Cook 2003) and Impromptu (Sorensen 2005).

Generative processes and music making

There is a considerable history of research into generative principles in music that we are building upon in our live coding practice. This includes investigations into generative systems in the fields of mathematics and music (Assayag et al. 2002), musical analysis (Lerdahl and Jackendoff 1983), algorithmic composition (Holtzman 1981), and the psychology of music (Sloboda 1988). In sympathy with research into musical mathematics, analysis, and algorithmic composition we are concerned here about the representation of algorithms and their affordances for musical control and output. With relation to the music psychology research we are interested in human engagement with the improvisational experience, interaction with the computer as instrument, and the generative opportunities that the musician brings to the live coding context through their domain knowledge and experience.

We have experimented with a range of generative process over many years (Brown 2002, 2004, 2005; Sorensen and Brown 2007, 2008) and it has become clear that a familiarity with the tendencies of various processes, gained through experience, and the careful combination of them to complement one another, are important to effective algorithmic music outcomes.

The demands of live coding require a focus on efficient methods of description and interaction. Even though many generative processes enable live coding in theory, there are practical and musicological reasons why not all classes of generative processes are suitable for live coding. Furthermore, we argue that live coding places unique demands on the operation and construction of generative musical algorithms due to the embedded nature of the human performer. An example may help illustrate our point about the affordances of algorithmic representation for live coding.

Below are two versions of an algorithm that play back a sequence of pitches as a ‘melody.’ The first algorithm uses a prepared function called ‘seq’ that takes as arguments;

- the instrument to use for playback,
- a list of pitches to play
- the dynamic value for note velocity,
- the duration for note length.

```
(seq piano (list 60 62 63 68 67))
```

The second algorithm below defines the ‘seq’ function and takes as one argument, a list of pitch values (the same as used in the first algorithm). The function is called by the last line of code.

```
(define (seq pitches)
   (au:play-note (now) piano (car pitches) 100 20000)
   (if (null? (cdr pitches))
       'done
       (callback (+ (now) 22050) 'seq (cdr pitches))))

(seq '(60 62 63 68 67))
```

Although the definition of the second algorithm is substantially more descriptively complex than just calling a pre-prepared function we are far more likely to use the full description in aa-cell performances. Our primary motivation for this decision is descriptive transparency - our ability to further interact with the algorithm at a descriptive level (i.e. at a syntactic level).

In the second algorithm the playback and looping (iteration) aspects typical of a sequencer are explicitly written and thus available for modification, whereas in the first algorithm these are hidden and so once the function is started the note stream cannot be modified. Other benefits of the second approach include that:

- it can produce a continuous note stream rather than one of fixed length
- it can be halted at any time
- variables can be changed on the fly
- control structures can be added and manipulated

Consider the following trivial extension which adds a counter melody, loops the sequence with pitch variation and applies a random choice of note duration. All of these modifications to the existing algorithm can be made at runtime because of our ability to modify the algorithm at the syntactic level.

```
(define (seq pitches)
   (au:play-note (now) piano (car pitches) 100 20000)
   (au:play-note (+ (now) 11025) clarinet
                 (pc:relative (car pitches) (random 2 5)
                 (pc:scale 0 'aeolian)) 60 20000)
   (if (null? (cdr pitches))
       (callback (+ (now) 22050) 'seq (jumble (list 60 62 63 68 67)))
       (callback (+ (now) (random '(22050 11025))) 'seq (cdr pitches))))
```

As the example above shows, the way in which an algorithm is represented can impact upon its utility for the live coder. This is because live coding is not a mere regurgitation of algorithms and their mapping to musical parameters. In live coding an algorithm is rarely constructed and then left to run its natural course, it is often modified as the performance proceeds. Given that the results of a generative process may not be completely predictable, interaction with them has the characteristics of improvisation. We have found in our performances similar tendencies to those observed by Eric Clarke (1988:5) around the generative nature of variation in instrumental performance where ‘The significant mismatch between projection [intention] and outcome lies in its effect upon the continuous construction of a generative representation during a performance.’ During live coding the human performer is a direct participant in the unfolding of the algorithm over time, not a mere bystander. It is an inherently interactive and expressive process. The performer is directly embedded within the algorithmic process and is free to guide and directly manipulate the unfolding of processes

over time. The generative process exists on two levels, the playing out of the algorithmic potential of the code and the unfolding of the algorithmic opportunities and structural pathways held in the mind of the performer. There are a range of algorithmic characteristics that we feel are pertinent to their successful application to live coding.

Complexity

The description length and complexity of an algorithm plays a large factor in its appropriateness for live coding. Algorithms such as neural networks, evolutionary algorithms, agent based systems, and analytic systems are all affected by issues of description complexity. The longer the description of the algorithm, the more time will pass writing the code in which the programmer is unable to pay attention to other aspects of a performance. Compounding this, a live coding performance is likely to consist of many concurrent algorithmic processes. Additionally, the complexity of the description is likely to restrict modification during a performance and contribute to the overall probability of error. In contrast to this, we consider algorithmic directness one of the most powerful aspects of live coding. In other words the human is directly in control of many of the temporal aspects of the algorithmic process and can change and modify the high level direction of a performance. Multiple performers can help to mitigate issues of complex description by cooperatively alternating their activities such that a suitable level of variation is always maintained. However, we would suggest that even in situations where multiple performers are working in concert, there are good reasons to limit the descriptive complexity of algorithms. In part our reasons for making this claim relate to a disruption of experiential "flow" that descriptively complex algorithms are likely to cause. We will discuss flow more fully below.

Abstraction

Related to the issue of description complexity is abstraction. Abstraction is necessary when working with digital computers because there is no possible way for us to deal with the complexity of the underlying operating system and hardware without levels of abstraction. However, this does not diminish the fact that software abstraction makes a substantive difference to the ability to express ideas, in both a positive and negative way (Abelson and Sussman 1996). Each live coding performer will make different decisions about appropriate use of code abstraction depending upon the focus of their activity and the context of their performance. As performers understand the types of interactions that are most significant to their practice it should become clearer where the use of library code is necessary and when transparency is more valuable. It is important to recognise that these decisions are not primarily about technical ability or audience perception, but are fundamentally related to the ability to express ideas in code. When programmers make a decision to abstract code away into a library, an abstract entity which can only be accessed as a 'black-box', the ramification is that they no longer have the ability to directly manipulate the algorithmic description.

Efficiency

The execution speed of an algorithm is, fairly obviously, of concern to live coding practitioners. Limits on computing resources may preclude the use of many types of searching, analytic, and complex rule based algorithms. There are a number of

demands on a computing system during live performance that need to be considered. These include audio and visual signal processing, text editing, event scheduling and execution of generative music algorithms. The balancing of these tasks within computing constraints requires awareness of the resource impact of each element and how they may interact. In particular those processes that maybe non real-time, such as genetic algorithm fitness selection, are difficult to incorporate and our experience is that their use invariably leads to less control, spontaneity, adaptability and fun.

Temporal scope
The temporal scope of an algorithm is also a consideration in live coding performance. By temporal scope we refer to both the amount of historical data required by an algorithm for analysis, what knowledge of future events are required, and how far into the future an algorithm must schedule events. Many grammars, pattern matching and analysis systems require a substantial amount of look-ahead for decision making and also often require the generation and scheduling of material into the medium to distant future. We have found these types of algorithms to be not very valuable in practice as they limit our ability to affectively respond to other concurrent processes, input devices and, most importantly, fellow performers.

Mapping
It is theoretically possible to map the output of any generative process onto a musical or sonic parameter, for example it is common to map a periodic function to a filter cutoff in a synthesis process and we often map periodic functions to note dynamics to create accented and pulsed sequences. However, if we wish to map a generative process onto a musical process, not merely a single parameter, the list of applicable generative processes is vastly reduced because tight correlations between mathematical functions and 'musical' patterns are limited. Furthermore, in a performance practice which emphasises the expressive affordances of program code we argue that the directness of the mapping is also of primary concern.

The relatively direct mappings that we have found useful for describing musical ideas in our live coding practice are amendable to the incorporation of common musical practices. We have found our musical domain knowledge of far more direct relevance when live coding than in our previous electronic music practices. We are also rediscovering a level of symbolic musical engagement that has been rapidly disappearing from the musical landscape.

While the current discussion may seem to severely limit the generative processes suitable for live coding, we have identified a set of algorithms that we have found particularly valuable (Sorensen and Brown 2007, 2008). These include, probability, linear and higher order polynomials, periodic functions and modular arithmetic, set and graph theory, and recursion and iteration. We use this set as the building blocks of our practice and they have proven robust in combination and applicable to a surprising variety of musical styles and contexts. For examples of these in use see http://impromptu.moso.com.au/gallery.html.

Performer engagement with generative processes

The role of the performer in live coding is fundamental to its importance as a mechanism for creative constructionsim (Papert 1993). Being embedded in the centre of the algorithmic process makes live coding an engaging experience and leads to effective construction of digital media. What do we mean by embedded? Ideally, we are suggesting that performers interact with algorithms that are designed in such a way as to give the performer intimate (transparent) and immediate (real-time) control over their execution.

With all of the limitations we have outlined about the choice of generative algorithms for live coding it might seem that the practice may become stifling. Our experience is exactly the reverse: the constraints that we have outlined focus precious cognitive and time resources on generative processes that are flexible and efficient. This leads to a musical practice that balances the capabilities of the computer and the human. Here are a few of the reasons why we believe this to be the case:

Engagement
Live coding promotes creative exploration and encourages 'flow' (Csikszentmihalyi 1992). The theory of flow, or optimal experience, highlights the need to balance skills and challenges in order to maximise the pleasure derived from creative tasks. Live coding requires a reasonably high level of skill when compared to other live electronic music performance practices, but it provides a correspondingly flexible vehicle for the musician to develop their intellectual, technical, compositional, and performance abilities and to balance these with appropriate musical challenges to achieve a considerable level of virtuosity. As a creative task, live coding incorporates knowledge of composition, improvisation, musicianship and computer science and therefore provides many dimensions for exploration and discovery.

Our discussion of flow through live coding may seem counterintuitive to many who see programming as a cognitive activity unsuited to intuitive performance. Our experience suggests that programming can become intuitive. Ideally programming becomes a means for musical expression and an extension of musical imagination. Of course the engagement with music making that results in flow can be broken by technical failures such as programming bugs, poor algorithm selection, inadequate tools, and so on, but this is not a deficiency of live coding as a practice but of our immaturity as practitioners and tool makers. We suggest that while digital creative tools are still underdeveloped it is still possible through practice to develop an intimacy with existing tools that leads to sustained flow experiences (Brown 2000). Of course we can always build better tools, but this should not stop us from learning to express with the ones that we have. This often means working around the deficiencies of the technology, just as baroque trumpet players made extensive use of the difficult higher registers of the instrument in order to circumvent a lack of valves.

Control
There is a well-established understanding from musical aesthetics (Kivy 2002) and psychology (Meyer 1956) that effective music requires a balance between simplicity and complexity. It can be difficult to find generative processes that produce an appropriate level or range of interesting output. Many simple

processes, such as repetition, can become tedious, while others, such as randomness, can seem featureless and uninteresting. Here we are not talking simply about signal entropy but expectation as a cultural characteristic. We would agree with music theoreticians that balancing cultural norms and deviations is vitally important to successful musical engagement (Meyer 1956, Kivy 2002, Huron 2006). It would, therefore, appear appropriate to consider control as an important ingredient in the development of musical algorithms. This balancing of control and surprise is a constant challenge for generative sound artists and our experience suggests that at present it is better handled by the performer than by some computational 'agent.'

Responsibility

Many of the more challenging issues related to algorithmic composition, such as high level structure and form, are mitigated in live coding by the direct intervention of the human performer. In a very real sense this human guidance is at the core of live coding, and we believe makes it a substantial departure from existing approaches to algorithmic and generative music. In our live coding practice computers provide low level and mid level pattern making capacity and humans provide the overriding cultural and contextual understanding required for a meaningful musical outcome. This shared responsibility for the musical outcome provides for a rich partnership between performer and instrument.

Code creation

While we have warned against use of prepared algorithms as 'block box' generators, we have found that code generators have been an efficient way to save time in live coding performances. As a result we routinely use macros to generate algorithm templates for common programming 'patterns' that we can fill out as required. We have also explored the integration of genetic programming techniques directly into the live coding environment. Genetic Programming, in which a programming language parse tree is constructed using genetic programming techniques such as mutation, cross-over and competition, can be used to manipulate existing or generate new code assets. This technique can be used in live coding environments not only at the runtime level but also at the syntactic level where 'evolutionary' changes to user-developed coding assets can be made in-place within the text-editing environment.

Efficiency

Finally, live coding is fun and productive. The amount of music that we have been writing since beginning our exploration of live coding is many factors greater than the sum of our algorithmic works in the many years prior to our commencing live coding. Note here that we say writing, and not simply performing. Through live coding with generative processes we are constantly creating new works, often in vastly different styles and contexts. We can think of no better marker of success than this dramatic increase in our music making and performance schedule.

Conclusion

Live coding with generative algorithms is a practice that embodies improvisational and compositional elements with a balance of directed and automated music making. We have suggested that generative algorithms for live coding should be succinct, have wide applicability, be computationally efficient, and require limited

temporal scope. These characteristics acknowledge the fact that a human performer is engaged with directing their use and evolution during performance and can pay attention to the large-scale structure and the cultural appropriateness of a work. We have situated our live coding practice within the wider context of generative music systems, examined the features of tools and technologies that support the work, and have discussed various attributes of generative processes and ways of interacting with them that suit live coding. These insights have assisted us to make our music making more creative and productive and we hope that they may do the same for many others.

Acknowledgments
The authors of this paper acknowledge the support of the Australasian CRC for Interaction Design (ACID) through the Cooperative Research Centre Program of the Australian Government's Department of Innovation, Industry, Science and Research.

References

Abelson, H. and Sussman, G. J. (1996) *Structure and Interpretation of Computer Programs*. Cambridge, MA, The MIT Press.

Assayag, G., Feichtinger, H. G. and Rodrigues, J. F., Eds. (2002) *Mathematics and Music*. A Diderot Mathematical Forum. Berlin, Springer.

Brown, A. R. (2000) “Modes of Compositional Engagement”. Proceedings of the *Interfaces: The Australasian Computer Music Conference*, Brisbane. The Australian Computer Music Association, pp. 8-17.

Brown, A. R. (2002) “Opportunities for Evolutionary Music Composition”. Proceedings of the *Australasian Computer Music Conference*, Melbourne. ACMA, pp. 27-34.

Brown, A. R. (2004) “An aesthetic comparison of rule-based and genetic algorithms for generating melodies”. *Organised Sound*, 9(2): 191-198.

Brown, A. R. (2005) “Generative Music in Live Performance”. Proceedings of the *Australasian Computer Music Conference*, Brisbane, Australia. ACMA, pp. 23-26.

Brown, A. R. "Code Jamming." *M/C Journal 9.6* (2006) 13 Jan. 2007 <http://journal.media-culture.org.au/0612/03-brown.php>.

Clarke, E. (1988) “Generative principles in music performance”. In, *Generative processes in music: The psychology of performance, improvisation, and composition*. J. A. Sloboda, Ed. Oxford, Oxford University Press**:** 1-26.

Collins, N., McLean, A., Rohrhuber, J. and Ward, A. (2003) “Live Coding in Laptop Performance”. *Organised Sound*, 8(3): 321-330.

Csikszentmihalyi, M. (1992) *Flow: The psychology of happiness*. London, Rider Books.

Dean, R. (2003) *Hyperimprovisation: Computer-Interactive Sound Improvisation*. Middleton, A-R Editions.

Holtzman, S. R. (1981) “Using Generative Grammars for Music Composition”. *Computer Music Journal*, 5(1): 51-64.

Huron, D. (2006) *Sweet Anticipation: Music and the psychology of expectation*. Cambridge, MA, The MIT Press.

Kivy, P. (2002) *Introduction to a philosophy of music*. Oxford, Oxford University Press.

Lerdahl, F. and R. Jackendoff. (1983) *A generative theory of tonal music*. Cambridge, Mass: MIT Press.

Loy, G. and Abbott, C. (1985) “Programming languages for computer music synthesis, performance, and composition”. *ACM Computing Surveys*, 17(2): 235-265.

McCartney, J. (1996) “SuperCollider: A new real-time sound synthesis language”. Proceedings of the *International Computer Music Conference*, San Francisco. International Computer Music Association, pp. 257-258.

McLean, A. (2004) “Hacking Perl in Nightclubs”. *perl.com* O'Reilly. Retrieved 1 May 2005, from http://www.perl.com/pub/a/2004/08/31/livecode.html.

Meyer, L. B. (1956) *Emotion and meaning in music*. Chicago, The University of Chicago Press.

Papert, S. (1993) *The Children's Machine: Rethinking school in the age of the computer*. New York, Basic Books.

Pearce, M., Meredith, D. and Wiggins, G. (2002) “Motivations and Methodologies for Automation of the Compositional Process”. *Musicae Scientiae*, 6(2): 119-147.

Schön, D. A. (1987) *Educating the Reflective Practitioner*. San Francisco, Jossey-Bass Inc.

Sloboda, J. A. (1988) *Generative Processes in Music: The psychology of performance, improvisation and composition*. Oxford, Clarendon Press.

Sorensen, A. (2005) “Impromptu: An interactive programming environment for composition and performance”. Proceedings of the *Australasian Computer Music Conference 2005*, Brisbane. ACMA, pp. 149-153.

Sorensen, A. and Brown, A. R. (2007) “aa-cell in practice: an approach to musical live coding”. Proceedings of the *International Computer Music Conference*, Copenhagen. ICMA, pp. 292-299.

Sorensen, A. and Brown, A. R. (2008) “A Computational Model For The Generation Of Orchestral Music In The Germanic Symphonic Tradition: A progress report”. Proceedings of the *Sound:Space - The Australasian Computer Music Conference*, Sydney. ACMA, pp. 78-84.

Wang, G. and Cook, P. R. (2003) “ChucK: A Concurrent, On-the-fly, Audio Programming Language”. Proceedings of the *International Computer Music Conference*. ICMA, pp. 219-226.

Whitelaw, M. (2004) *Metacreation: Art and Artificial Life*. Cambridge Massachusetts, The MIT Press.