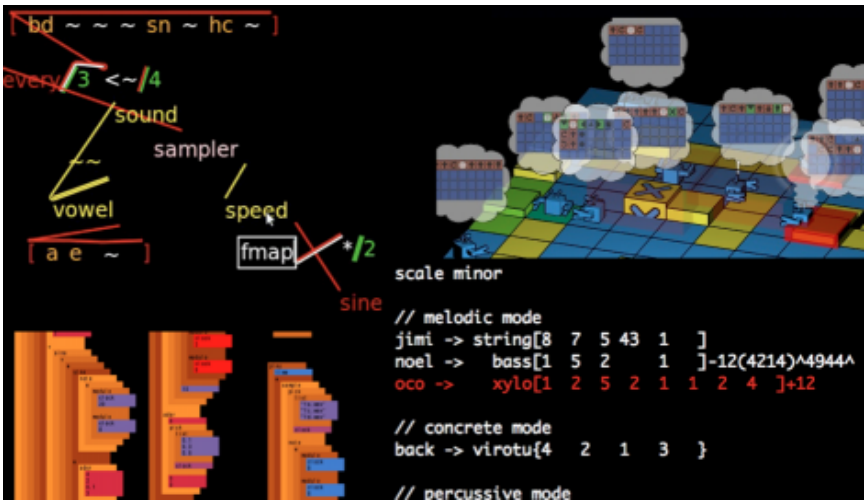
THE MUSICAL SCORE: THE SYSTEM AND THE INTERPRETER

This paper introduces live coding as a new path in the evolution of the musical score. Whilst being the perfect vehicle for the performance of algorithmic music, it also trans forms the compositional process itself into a live event, where play and generativity be come essential. Live coding is presented as a highly technologized artistic practice, often embracing graphical elements and language syntaxes foreign to standard programming languages.

AUTHOR(S)

- Thor Magnusson



It is a fact that in the evolution of instrumental music the performing musician has been condemned more and more to converting increasingly complicated scores into tones. Musicians became a sort of machine substitute, and finally there no longer remained any room for "free decision," for interpretation in the best sense of the word. [10]

OPENING THE MUSICAL SCORE

The musical score has always been in constant evolution. The score implements the idea of encoding music such that it can be stored, disseminated, and performed at later occasions. Initially the score served as recording device, but it quickly be came a tool for thinking music at a more complex level, serving as an extension of the composer's cognitive capacity. The idea of looking at the score as a technology that has had dramatic effects on music corroborates with the findings of schol ars who study the difference between oral and literal cultures [9]. Concomitantly, any history of the musical score will demonstrate the increase of sophistication in the control composers assert over performers. Whereas in early medieval and renaissance scores large parts of the music were improvisations, the 20th century score became highly specific, where explicit and increasingly nuanced instructions were given about pitch, loudness, rhythm and articulation.

In the late 19th century, systems using loom punch cards in order to encode music for machine playback (e.g., in pianolas and other musical automata) became popular and this development of automatic music took a leap in the 20th century with the commercial use of audio recording. As a response, and in an opposing spirit to this machine representation of music, composers began exploring new systems for composition, where graphical scores and textual instructions became more prominent. Related to these developments, the score was increasingly understood as describing gestural information for the performer, rather than being descriptions of pitch organised in time. The works composed in the first half of the 20th century for prepared piano, e.g., Villa-Lobos's *Chorus no. 8* (1925) or John Cage's *Bacchanale* (1940), are good exam ples of this shift.

A further evolution took place in the 20th century's conception of the musical score. With works such as Stockhausen's *Plus-Minus* (1963), Pouissieur's *Scambi* (1957), or Cardew's *Treatise* (1963-67), we gain the notion of what Umberto Eco calls the "open work" and the "work in movement", where the form of the composition itself "consists of unplanned or physically incomplete structural units" [5]. In these pieces performers are given a chance to improvise and interpret the score, such that the work has much wider expressive scope. In the words of Stockhausen from the same text as the initial quote:

But fit is noteworthy that the same composers who had called electronic music to life, *parallel* to this work in the years since 1956-57, published compositions which present the performing musician with a completely new responsibility. [...]

in this new instrumental music the performer is granted fields for free, spontaneous decisions, to which machines are not amenable. [10]

The motive of this article is not to analyse how machines have evolved since Stockhausen wrote this text. What is of inter est is how, in Eco's definition of the open work, it is clear that the openness under discussion is not a passive interpreta tion of the piece, but rather an active engagement where the structural elements of the work are manipulated by the per former or the listener. The open work is therefore a system that enables the interpreter to actively engage with the score it self, reinterpret it and appropriate it to the context in which it is performed. In a way, a feedback loop has been intro duced between the performer and the score; an idea that (strongly amplified) becomes a central feature of live coding per formances.

The musical score is a system of instructions: abstractions that represent musical gestures for the interpreter of the score. From this perspective it does not differ much from the instructions inscribed in software that generates music. In fact, there is no intrinsic difference in writing music in the traditional staff notation or as software code. The question is rather what kind of interpreter the music is written for and what logic is applied for non-deterministic decisions, if those are used in the piece. As a matter of fact, the computer can easily read staff notation and the human instrumentalist can per form music by reading computer code. This paper will look further at the question of the interpreter, but also how live cod ing systems, particularly as they emphasise the performance elements of musical coding, can be seen as musical composi tions in their own right.

INTERPRETING CODE: THE APPEAL OF SOFTWARE ART

In the last decade software art has become a prominent field where musicians, designers, programmers, and artists of var ious art forms explore the potential of the computer to execute their art. Traditionally, software art has focused both on the formal structure of code, following the modernist tradition, and engaged with society and politics where software serves as a cultural critique [4]. Since its inception, software art has shown that writing code can be just as gratifying artistically as painting on canvas or dancing on stage (see www. runme. org). Indeed, the ubiquity of computational algorithms in our daily life has given birth to a new research field called software studies, which engages with the role of software from a cul tural theoretical perspective [6]. Software art might be a subordinate field of this research programme, but the question remains if the category of software art is merely a transitional one, due to the omnipresence of software in all types of art and the general realisation by artists that generative features can add value to their work.

Artists have become attracted to software. In the art after Fluxus – which emphasised process, audience participation, and performance as algorithmic instructions – practitioners have become intrigued by the potential of delegating some of their creative decisions to algorithmic processes, which can be based on, for example, stochastic models, environmental variables (such as weather, animal behaviour or human interaction), or artificial intelligence and artificial life. Computa tional creativity, the idea of writing systems that produce novel solutions to 'problems' such as creating art, has proven ap pealing [2]. In this field, various generative music systems have been created with impressive results, and in the world of painting one might consider the work of Harold Cohen, with his robotic painter Aaron, as a strong candidate (http:// crca. ucsd. edu/ ~hcohen/). More recently, this idea of creative delegation to non-human processes is strongly explored in bio- art where biological systems are used as parts of the aesthetic process that create, or indeed *are,* the artwork. (see for ex ample the work of Andy Gracie at www. hostprods. com or the activities of Symbiotica at www. symbiotica. uwa. edu. au)

Although software art is often exhibited in galleries, its natural habitat is on the internet where software can run in the user's browser or be downloaded as standalone applications. Much software art retrieves data from online servers, strengthening the dependency on the networked grid and relating strongly to another strand of computer based artistic practice called net art [1]. The software is an artistic product, like a book or a film, to be enjoyed and interacted with by the user. It is a system of abstractions that can present any artistic media, but at the core is the artistic expression whose *material* can be any medium, multimedia or code itself, but whose *form* is necessarily encoded as software code. Often the software is of a generative nature, rendering new versions of the piece, each time it is executed. Here the software op erates as a score or a script that is interpreted by the machine. Indeed, in many programming languages, the primary sys tem that executes the code is called the *interpreter,* resonating with how musical scores can be seen as abstract instruc tions performed by the human interpreter, the instrumentalist.

LIVE CODING AS DELEGATION OF INTENTION

Musical live coding [3, 11], arguably a subordinate field of software art, is different in this sense as it requires the human to perform the work. In order to be able to write code as a performance act, the programming language has to be at a level allowing the live coder to quickly write the musical algorithms in an improvisational way. Few audience members would endure the manner in which compiled languages are written, for example. This has resulted in a practice where the live coder "composes" a system outlining beforehand the musical constraints. This, in many ways, is akin to composing music, in particular the 20th century open work scores (as analysed by Eco) that typically break out of traditional notation, for ex ample by using graphical encodings. Creating a live coding system can therefore be seen as a compositional activity al though, of course, the boundary between a composition and an instrument are never clear in computer music.

In live coding, the system can be seen as carrying out the role of the musical score, performed by the live coder. This per formance is based on a strong practice of engaging with the score by writing instructions for the *language* interpreter. We therefore have a twofold layer of interpreting, one where the human interprets the piece (which here is seen as the live coding language itself) and another where the language interpreter interprets the human performance. Between the live coding system and the interpretation of the computer, a space is created in which the live coder improvises, composing in

real-time through writing code, in a style of performance that requires a journey back to the baroque period if we want to find a strong parallel (an activity then called *extemporisation*).

From this perspective, one can frame the live coder as a delegator akin to the artificial life or biological systems mentioned above. If the live coding system is seen as a musical piece itself, the performing live coder becomes the interpreter of that piece, rendering variations of its generative potential.

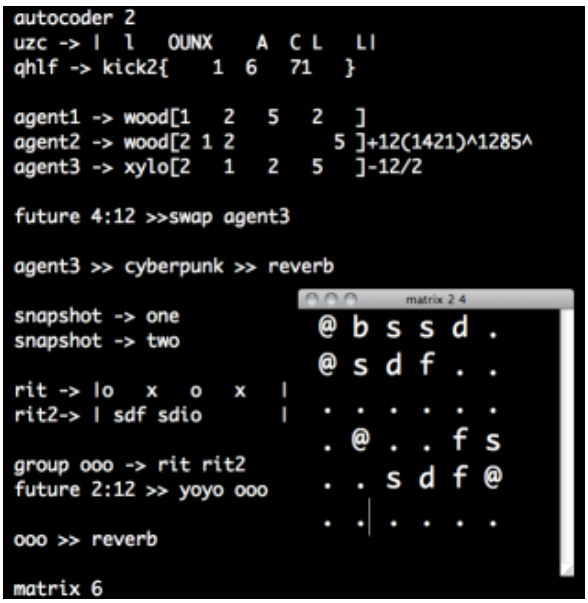
It should be noted that live coding does not have to involve computers at all. Nick Collins has, for example, done various ex periments with live coding of human performances where he has worked with choreographers, dancers and music impro visers in writing instructions that can be algorithmically executed. Collins defines live coding as an activity that necessar ily includes a reflexive element, stating that the "more profound live coding must confront the running algorithm" [3]. If we accept this strong definition of live coding – and Collis does acknowledge that many live coding performances do not live up to this promise – it is clear that live coding introduces a new form in musical practice that actually requires per formers to exercise their free will during performance and rewrite the score (the running algorithm) ad hoc.

THREE LIVE CODING SYSTEMS

The live coding systems I have in mind in this article are typically custom made and unique expressions of their authors. Although one can live code with any interpreted language, such as Python or Scheme, the typical live coding system is built *upon* such a language, creating a higher-level abstraction that effectively becomes the scope in which the performer works. The questions of affordances and constraints in creative systems become pertinent here, since the constraints of a system often yield strongly creative output [8].

Alex McLean is currently working on a live coding system called Texture where one writes textual instructions for the inter preter, but this text is of spatial nature where location and proximity define the meaning of the words and their functions within the main synthesis or musical graph. The aesthetics of this programming language reminds us of concrete poetry, whereas the functionality resembles a mixture of Pure Data and SuperCollider, which are both audio programming lan guages used in live coding. The environment is impregnated with common computer music terms of oscillators and fil ters, and mathematical functions are presented with common symbols in addition to lines that define the relationships be tween the signal graph objects.

My own ixi lang is a highly constraint system built on top of SuperCollider. The idea here is to create a performance frame work enabling me to write quick code that results in the formation in a musical piece in few seconds. The more I play with this system, the more I get the feeling that the system itself is a musical score that I play differently each time I perform it. If ixi lang is a piece of work in itself, a performance using it becomes like a variation of its expressive potential. I should note that, unlike Adrian Ward with his Auto Illustrator software art piece (www. signwave. com), I do not consider myself a co-author of the music made with ixi lang. As a musical score, the ixi lang exist at a meta-level above direct instructions. The software was released two years ago and it has been very inspiring to see how other people use the system. Surveys, user testing and interviews have confirmed that users of this system enjoy the inbuilt constraints and report that this can yield creative results [7].



Nick Collins, a long time live coder, has recently created a live coding app for the iOS operating system. The application has a simple user interface, consisting of the letters in the TOPLAP (an organisation concerned with live coding - see toplap. org) name that can be dragged around the screen to program a virtual machine for sound synthesis. The patterns of these letters become the instruction set for the interpreter to generate audio, typically a quite noisy but interesting out put. This app is a good example of an open work in the form of a software, presented as a live coding system where the end-user performs the piece, in a manner envisioned by Pousseur when he composed his work Scambi. This work also

shows the potential of creating live coding systems aimed for distribution and active interpretation by the performer/listener.

It has to be noted though that live coding systems are very diverse, from high level languages to lower level systems, from graphical representation of code to textual, and using diverse types of input devices, although the computer keyboard might be the most common device. The definition in this paper of live coding systems presented as types of musical scores might be less appropriate for more general audio programming frameworks like Impromptu, ChucK or SuperCollider. The three systems discussed in this article should be seen as open work scores existing at the meta-level, thus giving the performer a wide scope for expressive freedom.

CONCLUSION

This article has focused on live coding systems as a development in the history of the musical score. The live coder has to practise the important skills, but the system enables the performance to be highly improvisational, in ways that relate back to medieval and baroque music, but contrary to how the late 19th and 20th century music tradition formalised the musical score and performance (with few exceptions). The paper has drawn these parallels with older music and systems of representing it, but also contextualised live coding as a practice that relates to new types of musical scores originating from the mid 20th century. Additionally, the technological framing of these types of works, strongly engaging with the history of machine music and its encodings, afford new ways of relating to the performer, the listener and the social setting in which musical pieces are presented. Live coding is therefore an offspring of two strong traditions involving the formalisation and encoding of music, often for machine realisation, on the one hand, and the open work resisting traditional forms of encoding, on the other.

Generative (or algorithmic) music is currently undergoing renaissance with novel live coding practices and explorations of algorithms as part of the digital musical instrument design. There is wider understanding of the fact that when our media become processor based, we are in an ideal position to write non-linear scores for new instruments that can contain generative algorithms as core of their sound engine. The days are gone when music has to be encoded or recorded onto linear formats. When the mp3 has been superseded by the app, prolific opportunities present themselves for writing novel music, including musical scores.

References and Notes:

1. Tilman Baumgärtel, *net. art 2.0* (Nürnberg: Verlag für Moderne Kunst, 2001)
2. Margret A. Boden, *Creativity & Art*, (Oxford: Oxford Univ Press, 2011)
3. Nick Collins, "Live Coding of Consequence," *Leonardo* 44, no. 3 (2011): 207-211
4. Florian Cramer, "Concepts, Notations, Software, Art," in *Read_Me* (Moskow: Macros-center, 2002)
5. Umberto Eco, "The Poetics of the Open Work," in *The Open Work*, (Cambridge: Harvard Univ Press, 1989): 12
6. Matthew Fuller, ed. *Software Studies: A Lexicon* (Cambridge: MIT Press, 2008)
7. Thor Magnusson, "ixi lang: A SuperCollider Parasite for Live Coding," *Proceedings of ICMC.* (Huddersfield, UK: Huddersfield University, 2011)
8. Thor Magnusson, "Designing Constraints," *Computer Music Journal*, 34, no. 4. (2010): 62-73
9. Walter J. Ong, *Orality and Literacy* (London: Routledge, 1982)
10. Karlheinz Stockhausen, "Electronic and Instrumental Music," in *Audio Culture: Readings in Modern Music*, eds. Christoph Cox and Daniel Warner (New York: Continuum Books, 2004): 378
11. Adrian Ward, Julian Rohrhuber, Fredrik Olofsson, Alex McLean, Dave Griffiths, Nick Collins, and Amy Alexander, "Live algorithm programming and a temporary organisation for its promotion," in *read me — Software Art and Cultures*, eds. Olga Goriunova and Alexei Shulgin. (Aarhus: Aarhus University Press, 2004): 242-262

[http://isea2011.sabanciuniv.edu/paper/musical-score-system-and-interpreter]