Proceedings of the 7th International Conference on Live Coding (ICLC 2023)
April 19th - 23rd – Utrecht, Netherlands

Be Brief: Convergences and Possibilities of Live-Coding and sctweeting

Fellipe M. Martins
Universidade Federal de Minas Gerais (UFMG)
fmartins@ufmg.br

José Henrique Padovani
Universidade Federal de Minas Gerais (UFMG)
jhp@ufmg.br

ABSTRACT

With the advent of real-time computer music programming languages, specialized practices have arisen. Live coding (LC) and SCTweeting (SCT) are significant examples of creative constraint-coding approaches to computer music. The former consists of improvised live-electronic music in which performers project their coding screen. The latter is the practice of code golfing for creating (and sharing on social media) 140-character codes that generate complex sounds. In this paper we seek to approximate these two, drawing a comparative conceptualization and outlining possible artistic and contextual bonds between them. We analyze how both scenes, being new forms of artistic practices, propose radical and non-traditional approaches to coding. For that, LC and SCT are associated with the concepts and theoretical elaborations of cybernetics and mechanology, such as *feedback loops* as well as the *isodynamism* between technical behavior and human thought. Thereupon, we discuss how constraints play an important role in LC and SCT to develop ideas and achieve idiosyncratic artistic results. Finally, we outline some possible interweaving and didactic potential that have been latent in both practices.

1 Introduction

In 2009, Dan Stowell, an academic researcher and developer of the audio programming environment SuperCollider (SC), began posting on Twitter fully functional SC code with the constraint of a 140-character limit. The tweets got the attention of the community. Rapidly, several people engaged in posting dense and intricate SC code within Twitter's 140-character limit. This communal atmosphere eventually led to the compilation of tweets into an album (Stowell et. al. 2009). To work within this character limit, the *SCTweeting* coders employed techniques such as using syntactic sugar, creative/heterodox digital signal processing, code reusability, among others strategies (see Figure 1). The resulting *SCTweets* were characterized by a variety of sonic outcomes, ranging from techno to noise music, often presenting some kind of programmer humor, textual trick, and double entendre.

```
play{a=SinOsc;Splay ar:a.ar(b=(1..7)/7,a.ar(b/77)+1*7**a.ar(777*b,77**a.ar(b+a.ar(b/77, a.ar(b)/7,77*a.ar(b/77,b,b*7)))))/2}//
#SuperCollider
```

Figure 1 – SCTweet *by Fredrik Olofsson (RedFrik) - tweet0323 - relying on the use of only two UGens and the overuse of the number 7 for producing a complex tonal-noise drone. Extracted from: http://sccode.org/1-4Qy*

While optimizing code for a limited amount of space may seem at odds with optimizing code for ease of typing (a common requirement in live coding performances), we aim to investigate the tensions and frictions between these practices and explore how the unique characteristics of each can aid the other. The objective is to bring together these two constrained based creative approaches to computer music, *live coding* (LC) and *SCTweeting* (SCT), and to conceptualize, compare, and contextualize their potential artistic and conceptual connections. Adopting Ratto (2011) strategy of critical making, a processes were people are encouraged to learn in/through the act of making instead of

focusing strongly in the result itself, we propose a discussion centered at the act of creating live performances with code as well as hyper condensing code for fitting the small 140-characters web shareable space.

The term *live coding* covers a broad range of practices. According to Blackwell et al (2022) the community prefers to adopt description instead of definition, some like: "writing software in real-time, changing a program while it is running, projecting the screen for the audience to participate in, writing as an improvisatory practice, composing live using textual notation, changing rules while following them, conversational programming (conversing with the computer in its own native language), thinking in public, and creating and using bespoke systems tailored for on-the-fly or just-in-time performance". Inherently to these definitions are the questions: what can be considered "live" in a laptop performance? Does exhaustive rehearsing risk messing up the true liveness of a performance?

2 Code Golfers and Live Coders

SCTweeting is rooted in a longer-term practice in computing circles called *code golfing*, which can be seen as a game/competition/recreational activity where the participants strive to solve computer problems using the shortest source possible (Wikipedia contributors 2022). This communitarian practice gave birth to complex, imaginative results that would barely be possible to come out of a traditional programming industry field: *compiler bombs*[1], *tweetable mathematical art*[2], and *make your language unusable*[3]. *Code golfing*, a practice implicating the optimization of code for the shortest possible length, often involves using *esoteric programming languages*[4], *obfuscation* techniques, *minification*, *retro computing*, *software art*, and *computer humor*. This results in disregarding the traditional computer science principles and industry standards in favor of code that creates a mind-bending experience for the user. In this context, users become innovators who challenge machine paradigms, programming languages become nonsensical collections of characters, programs become open-ended executions. Ultimately, the capitalist idea of *functionality* disappears in favor of a non-definable use of the device.

It is important to point out some similarities with the *live coding* community and practices. Foremost, recent computer music programming languages (Max, Pure Data, SuperCollider, etc) were not initially meant to be used on-the-fly, but to be used for live music with real-time capabilities. The possibility of programming while the program runs is a side effect that users turned into a new artistic practice (Collins et. al. 2003). In some computer music software, such as Max and Pure Data, we can still see remnants of the separation between edit and run modes. In SuperCollider, there is the need to manually initiate the run action. When coding happens as an artistic performance, it subverts the typical design goals of computer music software, pushing it to provide new tools for this scenario. LC performances, for instance, often require more than a basic terminal interface with real time capabilities. It is common to see the use of ASCII art, vintage computing visuals, and character-based graphical user interfaces to create a more interactive and engaging coding experience. These efforts aim to bring a more lively aspect to both the act of coding and the traditionally minimalist coding screen[5].

While both SCT and LC scene present a strong focus on electronic dance music, we would like to highlight some more radical examples in both fields, especially the ones that subvert the very concept of its name. For instance, *screenBashing* (2016) performance[6] by Magno Caliman, where multiple windows of processor-heavy programs displaying ascii animation runs until the laptop battery drains completely.

The same insubordinate and deviating approach towards computing can be seen in the SCT way of *code golfing*. We might point here an example by Juan Romero (Rukano), see Figure 2, where the class library is swept sonically no matter what can happen to the server or the ears of the curious player. In both examples, we see some influence of industrial software testing concepts like unit testing and stress/benchmarking testing, however intentionally decoupled from its innermost safety procedures or testing metrics.

```
a=UGen.subclasses;fork{loop{u=a.choose;try{play{u.ar*EnvGen.ar(Env.perc(0.1,1),doneAction:2)!2}};1.wait}}//#sc #supercollider
!!!NOT SAFE!!!
```

Figure 2 – SCTweet *by Juan Romero (Rukano). All UGen subclasses from the user system are sonically executed throughout the play method and using their default parameters, no matter if they are meant to produce sound or any other kind of data. Extracted from: https://sccode.org/1-4Qx*

[1] Small files which produce enormous output when interpreted by naïve software, see https://codegolf.stackexchange.com/questions/69189/build-a-compiler-bomb.

[2] 140 charters for the red, green, and blue values for a 1024x1024 image. See https://codegolf.stackexchange.com/questions/35569/tweetable-mathematical-art.

[3] Try to write some code in your language and make it not satisfying our criteria of being a programming language any more. See https://codegolf.stackexchange.com/questions/61115/make-your-language-unusable.

[4] We often see the usage of general purpose programming languages (GPL) on Twitter code golfing for generating visuals, for instance, javascript, P5.js, GLSL (https://www.shadertoy.com/) and HTML5/Canvas (https://www.dwitter.net/).

[5] For instance, Orca and Gibber, are popular live coding environments that made extensive use of these characteristics.

[6] see https://vimeo.com/148626379 and https://vimeo.com/212694246 .

3 Language, Mind and Code

LC and SCT are music creation practices that utilize computers as a technological mediation. By comparing these practices, we can explore the characteristics that distinguish them from other computer-mediated approaches to composition, performance, and improvisation using electroacoustic or real-time computer environments. Seen from this perspective, how do live coding and SCTweeting differ from other creative processes that utilize these same technologies?

Code lines, in both practices, do not just serve as a mere record of ideas or as the source code from which an application will be built or evaluated (Blackwell et al, 2022: 171-174). The process by which the code is created, run and shared assumes a performative aspect. While projected on screens during the performance or shared with ”followers”/”friends” in the social network timelines, these succinct lines of code do things as computational performative utterances (Austin, 1975). When displayed during a performance or shared on social media, these concise series of characters not only generate sounds and images, but also perform the artistic, technological, and social act that defines them as practices of ”live coding” or ”SCTweeting”.

In both practices, the act relies on the computer language and its particularities. The coder’s proficiency in these languages is essential for both LC and SCT. In LC, the coder’s ability to effectively communicate with the computer through code is crucial for successful improvisation and to prevent potential errors; in SCT, the use of concise and efficient code is necessary to work within the character length limitations. In either case, a common challenge stands out: to concisely express and elaborate creative ideas. Whereas some authors argue that we think in specific ways in different languages and that languages work as scaffoldings to the mind, computer languages used in LC, SCT, and code golfing seem to work in a similar way, enabling us not only to structure new ideas but also ”to reflect on our own thoughts and characters and our limited but genuine capacity to control and guide the shape and contents of our own thinking” (Clark, 2008: 44).

Unlike other computer music creation practices that rely on programming languages, the lines of code explored in algoraves or microblogging social networks must be concise, aesthetically pleasing, and effective, generating appealing sounds and/or images. In these contexts, computer languages serve not only as a means of assessing the coder’s ideas and creative strategies, but they also take on a structuring and literary role in a similar fashion to that which natural languages play in genres such as haikus, tankas, epigrams, aphorisms, and limericks. Like these literary forms, the content and expressive result of lines of code depend on both the writer’s fluency and style, as well as on the features of the language itself, including its syntactic possibilities and vocabulary of functions, classes, methods, patterns, and abbreviations. Coders often develop their own ”lyrical style” through the use of these resources, which can be heard or seen in the resulting sounds and visuals, but that also can be read as ”rhymes” and ”assonances” – regular patterns of code, recurrent parameters and variable values, recursions, and other repeating strategies that may serve as a recurring theme or as a technique to save and reduce coding time and code length.

4 Feedback Loops and Digital Objects

LC and SCT approach technology differently from traditional engineering standpoints by prioritizing the creative process and experimental approach over pragmatic functionality. If there is any problem to be solved, it is not exterior to coding itself as an expressive act or as a tricky puzzle. Although both LC and SCT employ tools and technical strategies typical of the programming field, their unique character – as a form of artistic experimentation – warrants a more nuanced conceptual approach. This approach can benefit from concepts and theoretical elaborations from fields such as Cybernetics and Science and Technology Studies. It is crucial to consider the specificities of our relationship with machines in creative practices and experiments such as LC and SCT: what are their practical, playful, and aesthetic purposes, and what can we learn from these interactions?

SCT and LC creations are not usually conceived to result in teleological goals. “Live coding is about disrupting the deterministic logic between notation and process, bringing it into a creative feedback loop…” (Blackwell et al, 2022: 131). The cybernetics concept of *feedback loops* or *chain of feedback* (Wiener, 2019[1948]: 131), highlights aspects of LC and SCT creative processes and the connections between humans and the technological tools at stake. The tools, in our case, include not only computers and electronic devices but also software, operating systems, programming languages, etc. While conceptualizing notating code, running the computational processes, projecting (or sharing) the results in the environment, and perceiving the sensory and social reverberations of his activity, the human-machine couple structures *feedback* processes. The multiple outputs of the system influence and affect the multiple inputs: the decision to create a new layer of LC patterns that may interfere with the previous ones, the adjustment of parameters in a SCT generating new unexpected results, new lines received from other coders recycling ideas previously shared, etc.

The relationship between the human mind and body and computer hardware and software is, thus, complex and multifaceted. It involves a feedback regulatory network that is composed of multiple inputs and outputs from both the human and technical aspects of the system. Olofsson (2020) draws attention to moments, while coding SCT, when the human mind seems to attempt to emulate a computer and vice versa. Olofsson refers to this as *cybernetic music in practice*. Blackwell et al. (2022) suggest that the ultimate goal of live coding is to achieve a cybernetic fusion between the human and the machine. They argue that, in this integration, the ability to anticipate the mechanisms and results of code is a necessary skill, and that this embodiment of code mechanics/outputs is a key aspect of live coding.

The TOPLAP ManifestoDraft also advocates for a self-regulating system composed of humans and machines (TOPLAP collective, 2005), aiming to include the audience in the cybernetic feedback loops. Its ambitious goals, including accessing the performer's mind and transcending artificial language, have significant ethical and political implications for topics such as free will, artificial intelligence, and human enhancement. Collins (2011) pinpoints a transition toward greater integration between humans and machines: "From software to wetware: Live coding itself raises consciousness of human beings within the innards of algorithms". He states that "live coding itself raises consciousness of human beings within the innards of algorithms". Recently, experiments involving biohacking and biotransducers have integrated the computer music reality and, to some extent, the live coding scene (Collins, 2011). As the possibility of the existence of a cybernetic being becomes more realistic, the ethical debate intensifies.

SCT and LC coders seem to be increasingly interested in using generative, machine learning, and artificial intelligence techniques to increase non-determinacy in their creative processes. This may involve granting the machine a greater *degree of freedom*. As a result, generative aesthetics, which vary in their level of human-machine interaction, tend to become an increasingly dominant trend in these practices, enhancing the complexity and intensity of the feedback loops at issue.

In addition to the cybernetic perspective, which focuses on feedback processes and the constraints that shape the course of events, it is also valuable to consider a *mechanological* approach to LC and SCT. *Cybernetics* examines the control and communication between humans/animals and machines. *Mechanology*, on the other hand, investigates how technical and digital objects *individuate as anthropological apparatuses,* and how humans invent and interact with them in psychophysiological terms. According to this view, these objects are the translation and deposition of human thoughts and gestures into machinic codes, gears, or other operational mechanisms.

While the *individuation* of technical/digital objects such as LC and SCT can certainly be further explored through the theories and concepts of Gilbert Simondon (2017[1958]) and Yuk Hui (2016), the concept of *isodynamism* between humans and machines appears to be more relevant for this current work. Simondon discusses the mental and psychological connection between the inventor (the coder in this case) and the machine (not just the electronic device, but also the combination of hardware, software, and technocultural network that enable practices like LC and SCT).

> The analogical relation between machine and man is not at the level of corporeal functioning; the machine neither nourishes itself nor perceives, nor rests, and cybernetic literature falsely exploits the appearance of analogy. The true analogical relation is in fact between the mental functioning of man and the physical functioning of the machine. (...) To invent is to make one's thought function as a machine might function, neither according to causality, which is too fragmentary, nor according to finality, which is too unitary, but according to the dynamism of lived functioning, grasped because it is produced, accompanied in its genesis. (...) One has to have invented or reinvented the machine if the machine's variations of functioning are to become information. The noise of an engine in itself does not have value as information; it takes on this value through its variation in rhythm, the change of its frequency or tone, the alteration of its transients which translate a modification of its functioning with respect to the functioning that results from invention. (Simondon, 2017[1958]: 151).

In fact, it is at this isodynamic level of making one's mind to "work like a machine" and, conversely, to make the machine "work like one's mind" that the processes of creation, transformation and projection/sharing of LC and SCT codes seem to happen. In the coupling between humans and machines that involves SCT and LC, there is a low level of alienation from coders concerning the technical/digital objects at stake. As Simondon points, this is not resumed to the ownership of the "means of production" (computer, software, audio devices, etc):

> The technical activity distinguishes itself from mere work, and from alienating work, in that technical activity comprises not only the use of the machine, but also a certain coefficient of attention to the technical functioning, maintenance, adjustment, and improvement of the machine, which continues the activity of invention and construction. (Simondon, 2017[1958]: 255).

5 Constrained Code as an Artistic Impulse

In addition to turning on feedback loops and auto-regulatory processes, cybernetics attempts to understand these relationships by considering that the course of events is subjected to *restraints* and *constraints*. These limitations and restrictions determine both the type interaction between humans/animals and machines, as well as the outcome of these cybernetic "systems".

> If we find a monkey striking a typewriter apparently at random but in fact writing meaningful prose, we shall look for restraints, either inside the monkey or inside the typewriter. Perhaps the monkey could not strike inappropriate letters; perhaps the type bars could not move if improperly struck; perhaps incorrect letters could not survive on the paper. (...) ...from the cybernetic point of view, a word in a sentence, or a letter within a word, or the anatomy of some part within an organism (...) are all to be (negatively) explained by an analysis of restraints. (Bateson, 1967: 29).

Even though performing musical LC and practicing SCT on Twitter do not share any particular apparent purposes, the artistic strategy in use shares a common bearing: constraining the use of a tool as a way to propel creativity toward novel artistic solutions. The process of writing the shortest text possible to express one deeply complex idea can be traced back to ancient records and genres of writing like the Japanese *haiku*, Greek aphorisms, philosophical fragments, etc. However, LC and SCT are not only brief: they are built to deal, briefly, with constraints. This strategy can be related to 20th-century literature movements like *OuLiPo*. The roots of live writing – or, more broadly, lively creation using words and spoken language – can also be traced to ancient forms like medieval troubadour improvisations, Brazilian *repente* declamation/improvisation duel, and late 20th-century battle-rap. Due to its demands for visually appealing text form, LC approach to displaying text can also be rooted in typewriting art, concrete poetry, and typography, besides the many other types of visuals used in performance.

It is relevant to analyze the dimension of gaming within LC and SCT. Both can be easily classified as a *game*, with the main difference being the time scale in which players take place. Interestingly, the constitutive aspect that generally defines a game is missing: acknowledgment of a winner as a means to end the match. Indeed, the quintessence of the artistic dimension favors plurality and avoids authoritative truths and rules. Moreover, in both LC and SCT, cheating becomes an honorable skill since it shows dexterity in the use of the programming language as well as promotes insights and excitement in the participants/audience.

Collins (2011) also relates elements of board and online games featuring structures highly similar to those of *live coding*. Adopting a broader definition, these features may also invite us to confabulate about the possibility of considering them large-scale political systems – in the sense of lively making and adequating a collection of laws – an example of LC out of the software/hardware realm.

In fact, it is important to highlight that artistic practices have used the idea of constraining specific features, deliberately or not, in order to achieve peculiarities in style definition. This seems to be the case here: while SCT is a conscious challenge of finding the shortest syntax that creates a sound piece, LC does not so deliberately impose to its performers the limitation of speed and fluidity which either the language or the coder must have to provide a convincing performance.

However, both LC and SCT have been criticized for their resulting limitations. *Live coding* is often criticized for sacrificing musical complexity and structure in favor of a fluid coding experience that avoids pre-composed blocks of code. Additionally, *live coders* and analog synth improvisers are often said to have too many parameters to control simultaneously, leading to slow-evolving performances. *SCTweeters*, on the other hand, are criticized for lacking musical/sound quality due to their size constraints, such as a lack of rhythmic construction in codes with impressive timbres or lack of interesting timbre when traditional melodic, harmonic and rhythmic content is complexly defined. Additionally, the limited length available in SCT can make it difficult to develop complex ideas when working with long built-in syntax, such as buffers or Phase Vocoder UGens

The central question that arises from these evaluations is: what is the goal of this specific artistic practice? In their chapter on this topic, Blackwell et al (2022) argue that live coding allows for the appropriation of algorithms for different purposes or even for the sake of the process itself. Ratto (2011) supports this idea, stating that, by constraining certain aspects of the creation process, it is possible to highlight aspects that are often overlooked when creators focus, excessively, on the outcomes.

As an example, Figure 3 presents a SCTweet that recreates the minimalist piece *Piano Phase* (1967). The piano sound is certainly not perfectly modeled, since it is the basic default SuperCollider synthesizer. The execution is also not challenging any human performer. Likewise, no spatial phase is reproduced here. What is then the purpose of doing such a tiny recreation?

```
f={|t|Pbind(\note,Pseq([-1,1,6,8,9,1,-1,8,6,1,9,8]+5,319),\dur,t)};Ptpar([0,f.(1/6),12,f.(0.1672)],1).play//#supercollider
reich RT @earslap
```

Figure 3 – SCTweet *by Batuhan Bozkurt and Charles Celeste Hutchins. The minimalist piece* Piano Phase *(1967) by the composer Steve Reich is reimplemented using SuperCollider under a maximum of length of 140 characters. Source: https://supercollider.github.io/sc-140.html*

Firstly, there is a typical programmer humor[7] involved in the challenge of recreating classical programs using every type of programming language, e.g. the snake or mario game being reprogrammed using programming languages designed for banking or for low level machine execution (8-Bit Show And Tell, 2019). Secondly, when we shift the support of an artistic creation, immediately several questions regarding its constructions and operation emerge. The process of remaking *Piano Phase* can lead to these questions through the recontextualization of the piece. Additionally, multiple programming paradigms can achieve the same result, causing us to consider which numerical approach is most effective, suitable, comfortable, or clear for expressing the intended idea. For example, in Figure 3, the authors encapsulate the piece into a function to call it again for the canon, highlighting the reusability aspect of the piece. Constraint rules can also lead to unique results, as demonstrated in Figure 4. SuperCollider has a limited set of built-in audio samples, including 4 seconds of baby talk and 2 seconds of low quality radio talk. The author has successfully rearranged the baby talk to simulate the speaking sound of the word "SuperCollider." Without these restrictions, would this collage approach have been discovered?

```
b=Buffer.read(s,"sounds/a11wlk01.wav");play{t=Impulse.kr(5);PlayBuf.ar(1,b,1,t,Demand.kr(t,0,Dseq(1e3*[103,41,162,15,141,52,124,190],4)))!2}
```

Figure 4 – SCTweet *by Nathaniel Virgo. Here one of the built-in audio samples (4 seconds of incomprehensible baby talk) is reassembled to sound like the spoken word "SuperCollider". Source: https://supercollider.github.io/sc-140.html*

Technological frameworks that are intended for artistic use are not neutral or transparent. While they may claim to offer a wide range of tools, this abundance of options can actually serve as a creative barrier. The excess of possibilities can become overwhelming, causing the user to lose focus and become lost in the midst of a vast array of tools. Paradoxically, deliberately imposing constraints on the vast possibilities offered by digital tools can actually help the artist to achieve a greater distance from the technological means and focus more on the artistic goal itself. Magno Calliman (2020) suggests experimenting with constraints in live coding as a way of exploring the possibilities and limitations of this practice: "the use of guitar as text input device; a live coding performance that only uses the mouse; programming without being able to see the computer screen and having to memorize the position elements in the code such as variables, functions and it's arguments; livecoding an Arduino synth that can physically catch on fire".

Furthermore, Olfosson (2013) goes in the same direction when proposing a live coding setup in which the transmission port of a microcontroller is hooked into an audio mixer and the live coder writes C programs to be transmitted to the serial port. Unusual factors like the baud rate, type of bootloader, number length, and program length count on the resulting sound. In this scenario, the live coder is sonifying the code itself or at least is a traveling information version. Despite the high abstraction of this scenario, Olfosson is still able to control high-level musical parameters such as timbre, noisiness, chords, and rhythms through very special cases where these aspects seem to happen.

In particular, a point that has been in debate for both LC and SCT is that it provokes in the audience an impression that the code itself (and its ideas) is more important than the artistic outcome. The importance of the poetics of the code itself is being raised to a new level with practices like LC and SCT: often an obscure syntax, superstitious numbers, or hacking style ASCII show engages the audience more than the sound itself. However, Collins (2011) evaluates that 'the more profound the live coding, the more a performer must confront the running algorithm, and the more significant the intervention in the works, the deeper the coding act', and the same is relevant in the context of SCT. Live coding and SCTweeting were born as attempts of dramatizing and animating a computer algorithm, recontextualizing the place where software was meant to be by means of giving visibility and movement to codes that were designed to be static texts.

6 Future Remarks: Possibilities and Risk

In the latter half of the 20th century, the field of algorithmic music saw a flurry of activity with the emergence and disappearance of various genres, styles, and practices. Live coding has managed to thrive as a technique and practice, regardless of the tools used. However, it remains to be seen if this is necessary for the longevity of the art form. While the SCTweeting movement may have slowed down somewhat, the practice of sharing small pieces of code for artistic purposes on social media is still very active, particularly in the creation of visual content using general purpose programming languages.

[7] One might say there is a small hidden joke in making minimalist music using minimalist code.

We shall emphasize that LC and SCT are processes and not results. They are neither musical genres nor types of organizing sound, but processes in which sound will be organized and, as result, some specific characteristics might arise thus defining a genre of style. As discussed by Magnusson (2019) generative art has long term problem with establishing a particular format or support media: artists who have strived to launch generative pieces under traditional supports (CD-ROMs, USB sticks, etc) saw a fast incompatibility of their products associated with a difficult spread. The same is true for digital formats, although on a smaller scale of incompatibility. Therefore, several generative practices have adopted the model of single performance, site specific or long/multiple rendered files for presentation, a partial solution for retaining the focus on the generative process. The main open question is still: what does the process of displaying code convey to the audience? Is it solely shifting audience attention from the musical act to the procedure of making music digitally? Or is it somehow instigating some Schaefferian manner of listening, where cause, mode and effect are to be perceived by the audience as a whole? If so, are we falling back on the dilemma of sacrificing hearing fruition for the sake of saving a conscious and rational sound perception?

Live coding gradually turns a blank screen into pulsating lines of codes that radiate melodies, rhythms, timbres. SCTweeting explodes complexity at a single keystroke, a one line program execution holding a musical piece to be analyzed and disentangled into its smallest constituents. Can these top-down and bottom-up approaches coexist into a single computer music practice? We believe that searching for ways to integrate both would increase the live dimensions of code golfing as well as expand the complexity of live coding results. SCT, in general, tends to generate live algorithms, which are not necessarily based on a natural or living system but display enough complexity to be perceived as autonomous and complex agents (Dean, 2018). In this sense, SCT cannot be easily used in the traditional context of LC, though welcome in the radical live coding practices in which intense risk is a desirable factor (playing with *SCtweets* can easily crash the server or destroy speakers).

Both LC and SCT offer great opportunities for developing anticipation and problem-solving skills. We also recognize that code plays a crucial role in facilitating communication and socialization within these practices. It is important to note that the success of SCT and LC relies on the sharing and exchange of ideas and feedback. Even if the code is not fully understood or error-free, the core process of these practices involves sharing and collaborating with others.

7 Acknowledgments

We would like to acknowledge CAPES and FAPEMIG.

References

Austin, J. L. 1975[1962]. “How to Do Things with Words”. Edited by J. O. Urmson and Marina Sbisà. Second edition. William James Lectures. Cambridge, Massachusetts: Harvard University Press.

Bateson, Gregory. 1967. “Cybernetic Explanation.” *American Behavioral Scientist* 10, no. 8: 29–29.

Blackwell, A. F., Cocker, E., Cox, G., McLean, A., & Magnusson, T. 2022. *Live coding: a user’s manual.* MIT Press.

Caliman, Magno. 2020. “Meditation and “object specific” creative processes in experimental live coding”. https://hybrid-livecode.pubpub.org/pub/magnocaliman/release/1

Clark, Andy. 2008. “Supersizing the mind: embodiment, action, and cognitive extension”. Oxford: Oxford University Press.

Collins, N., Mclean, A., Rohrhuber, J. & Ward, A. 2003. “Live Coding in Laptop Performance”. *Organised Sound* 8, no. 3: 321–30. doi:10.1017/S135577180300030X.

Collins, Nick. 2011. ”Live Coding of Consequence.” *Leonardo* 44, no. 3: 207-211. Available: muse.jhu.edu/article/431861.

Dean, Roger T. (2018). *The Oxford handbook of algorithmic music.* Oxford University Press.

Hui, Yuk. 2016. “On the Existence of Digital Objects”. *Electronic Mediations* 48. Minneapolis: University of Minnesota Press.

Olofsson, Fredrik. 2020. “Cybernetic music in practice”. *Ideas Sonicas* (v23 Jul-Dec 2020). Available: https://en.cmmas.com/sonicideas

Olofsson, Fredrik. 2013. *Arduino live coding.* Available: https://fredrikolofsson.com/f0blog/arduino-livecoding/

Ratto, Matt. 2011. *Critical Making: Conceptual and Material Studies in Technology and Social Life, The Information Society*, 27:4, 252-260.

Simondon, Gilbert. 2017[1958]. *On the Mode of Existence of Technical Objects.* Minneapolis, MN: Univocal Pub.

Stowell, D., Virgo, N. Bovermann, T., Walters, T. Bozkurt, B. Magnusson, T., Hoistman, C., Rohrhuber, J., Padovani, J., Dixon, J., Olofsson, F., Rutz, H. & Valle, A. 2009. SC-140. https://archive.org/details/sc140/

Wiener, Norbert. 2019[1948]. *Cybernetics: Or, Control and Communication in the Animal and the Machine.* Second edition, 2019 reissue. Cambridge, Massachusetts: The MIT Press.

TOPLAP collective. 2005. *TOPLAP Manifesto draft.* Available: https://toplap.org/wiki/ManifestoDraft. Last (accessed December 11, 2022).

Magnusson, Thor. 2019. *Sonic writing: technologies of material, symbolic, and signal inscriptions.* Bloomsbury Publishing USA.

Wikipedia contributors, "Code golf," *Wikipedia, The Free Encyclopedia,* https://en.wikipedia.org/w/index.php?title=Code_golf&oldid=1077599348 (accessed December 11, 2022).

8-Bit Show And Tell. *Super Mario Bros. on the Commodore 64/128D with SuperCPU.* April 19, 2019. Educational video, 15:22. https://www.youtube.com/watch?v=cLb0SDlvmfI.