LISTAHÁSKÓLI ÍSLANDS
Iceland University of the Arts

The end of Graphical Sequencers:

TidalCycles as a way of experimentation

Gustavo Nicolás Villanueva Cataldi

Final thesis for a BA-degree
Icelandic Academy of the Arts
Department of Music Composition – New Media
May 2022

The end of Graphical Sequencers:

TidalCycles as a way of experimentation

Gustavo Nicolás Villanueva Cataldi

Final thesis for a BA-degree in New Media Composition
Supervisor: Steindor Kristinsson

Icelandic Academy of the Arts
Department of Music Composition
May 2022

Abstract

As most musicians who have experience with digital tools already know, commercial Digital Audio Workstations (DAWs) and graphical sequencers, are rigid tools when it comes to experimenting with complex sequences and simultaneous parameter changes. Moreover, these environments silently influence the digital music composer's aesthetic decision and suggest specific working methodologies with their innate structure and interface design. Most of the time musicians will have to unconsciously alter their way of working for the ones suggested by the music software, embracing its conceptual and compositional constraints. this meaning that musicians will have their musical expression outlined by the piece of music software they utilize. This being said, there are tools that will shape the musician's mind-set in a more pronounced way than others.

The purpose of this essay is to compare Graphical Sequencers with text-based programming environments. I chose to analyze open text-based programming environments; thus, they allow a more extensive experimentation, simultaneous parameters control, complex sequence creation and sophisticated audio manipulation than graphical sequencers. In addition, text-based programming tools have no fixed user interface, the environments start up with a blank page a tabula rasa, having less influence over the musicians and allowing them more room for experimentation.

Among the wide scale of existing music programing tools, I will be mainly focusing my thesis on the analysis of TidalCycle as a case study. I will introduce the program basics, its inner architecture, how to structure and deal with pattern creation and also describe its approach to musical time.

Through this essay I'll examine the usability problem of graphical sequencers, the suggestion of technology on musician's workflow and the influence of graphical interfaces. Conceptual examples and use of actual code will be presented in this essay to illustrate particular cases. Finally, I will present the conclusions reached by the analysis of this subject.

Table of content

Introduction

Andrew Hugill describes live coding as "one of the purest forms of digital music"[1] and additionally states that it is the "ultimate objective of computer music"[2]. Live coding is the manipulation of computer code by the usage of particular programming languages and environments dedicated to composing music in real time. Live coding is composed by a wide range of programming tools and environments that stimulate the creation of complex sequences and audio manipulation. For example some of these coding programs are: : TidalCycles, SuperCollider, Max/MSP, Pure Data, ChucK, IxiLang, among others.

Thor Magnusson notes that these programming tools are categorized by "not only how they sound, but also by which musical patterning or form they afford"[3] and he also continues by adding that "specific musical environments encourage certain practices and prevent others"[4]. There are two particular formats of live coding environments, the simple textual user interface (TUI), represented in figure 1, and the more familiar and visual oriented Graphical User Interface (GUI) represented in Figure 2.

The more purist live coders might classify the GUI environments, like Max/MSP and its open-source substitute Pure Data, as live patching. This definition arises from Miller's Pucket original environment named "Patcher"[5], alongside and in remembrance of modular synthesizers, that used patch cables to interconnect between modules.

[1] Andrew Hugill, "Performance and Musicianship: Live Coding", in *The Digital Musician Second Edition* (New York: Routledge, 2012), 171.
[2] Thom Holms, "Early Computer Music (1953-85)", in *Electronic and Experimental Music: Technology, Music and Culture* (New York: Routledge, 2016), 293.
[3] Thor Magnusson, "Confessions of a Live Coder", in Proceedings of International Computer Music Conference (Huddersfield: University of Huddersfield, 2011), 1.
[4] Ibid.
[5] Miller Puckette, "The Patcher", in Proceedings of International Computer Music Conference (San Fransico: ICMA,1988), 420.

```
d1 $ n (snowball 4 (+) (slow (sine + 2.3)) "{0 [1 2] [3 4 5] 3 5 . 0 [3 4] 3}" |+ "<9 10 12>")
   # s "claus"
   # legato 30
   # hpf 250
   # room (range 0.3 0.5 sine) # sz (range 0.3 0.6 sine)
   # distort 0.75
   # gain 0.75

d2 $ n (snowball 4 (+) (slow (sine + 1)) "{0 [1 2] [3 4 5] 6 7 . 0 [3 4] 8}")
   # s "cpu2"
   # legato 2
   # distort 0.4
   # gain 1.15

d3 $ slow 2 $ n (snowball 4 (+) (fast (sine + 1)) "{0 [1 2] [3 4 5] 6 7 . 0 [3 4] 8}")
   # s "bsvocals"
   # legato 1
   # speed (slow 1.5 (range (-1) 2 sine + saw))
   # room (range 0.3 0.8 sine) # sz (fast 1.5 (range 0.3 0.4 sine + ("<0.5 0.6 0.4 0.3>")))


d1 silence
d2 silence
d3 silence

hush
```

Figure 1 – a screenshot of Atom text-editor

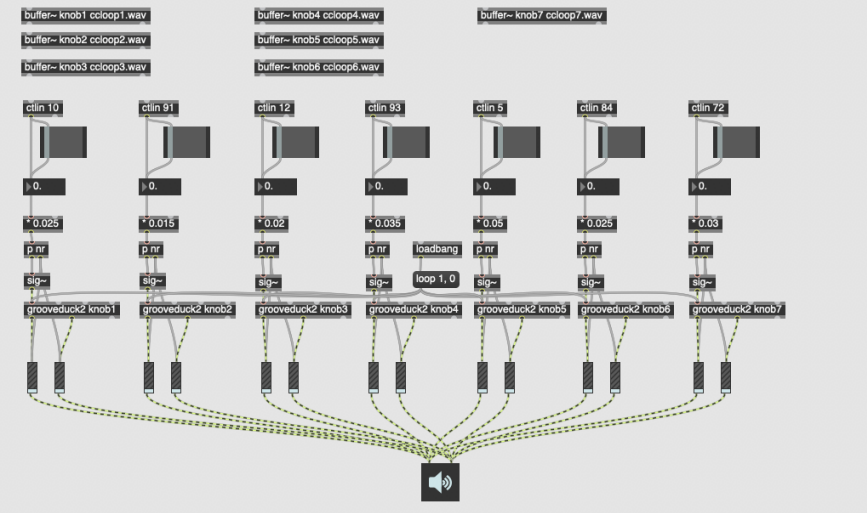

Figure 2 - a screenshot of a MAX patch

Live coding as a practice spur circa the year 2000 and it has been around ever since. Its beginnings can be tracked from different places, yet it seems it sparked as a reaction to the inflexibility and focused use of music software and Audio Workstations (DAWS), in conjunction to a comeback of "creative coding" as a pursuit of art form. Live coding has transformed into a phenomenon experienced all around the world, with a current variety of technologies, coding languages and live coding environments for the performing arts sphere.

Live music coding has developed into a lively worldwide community of researchers and coders over the last decade, engendering a new exploration domain, where computer science merges with the performing arts. In live coding, the source code is manipulated and edited in order to customize and rework a process that is already running. For the last decade, these live coding techniques have been progressively utilized as a medium for creating live music

improvisations[6], alongside new coding environments and programming languages developed as end-user music interfaces[7].

Live coding is in the present day, an active research space, a prominent topic in prominent computer music conferences, the study case of journal special issues[8], and the spotlight of international seminars[9]. Live coding has mainly matured from the digital performing arts scene and akin research spheres, although it is also associated with Computer Science sphere and Software Engineering, beneath the concept of live programming language investigation.

To recap, Live Coding is an activity where programmers write code for generating music, whereas an up running process is constantly being reworked and modified, without breaching out the musical output. The term "musical" is used here in a wide scope, because most of the time the results can get extremely complex and experimental, meeting the borders of what might be considered music.

As Alex McLean and Geraint Wiggins stated regarding live coding, "The archetypal live coding performance involves programmers writing code on stage, with their screens projected for an audience. The code is dynamically interpreted, taking on edits on-the fly without losing process state, so that no unwanted discontinuities in the output occur."[10]. This endeavor might happen by night within the context of a nightclub (e.g. as an Algorave)[11], or during daytime to a seated audience inside a concert hall (e.g. performance by laptop ensemble)[12], or in a more collective way, as a long form performance (e.g. slow coding)[13].

The live coder might be joined by other live coders or even by instrumental musicians, or perhaps by dancers and choreographers[14]. Anyway, the live coder would want to come into a state of focus and creative flow. There are several approaches to live coding music, one of the most popular approaches is based on an improvised jazz model. This meaning that none of the music is pre-composed, alternately all the music is composed through live experimentation.

---

[6] Nick Collins, A. McLean, J. Rohrhuber, and A. Ward "Live coding in laptop performance", *Organised Sound*, 8(03):322, 2003, http://dx.doi.org/10.1017/s135577180300030x.

[7] Alex McLean, "Making Programming Languages to Dance to: Live Coding with Tidal", in proceedings of the 2nd ACM SIGPLAN international workshop on Functional art, music, modeling & design (Gothenburg: Association for Computing Machinery, 2014), 63.

[8] A. McLean, J. Rohrhuber, and N. Collins, "Special issue on Live Coding: Editor's notes", Computer Music Journal, 38(01): 4, 2014, https://doi.org/10.1162/COMJ_e_00225.

[9] Blackwell, A. McLean, J. Noble, and J. Rohrhuber. "Collaboration and learning through live coding (Dagstuhl Seminar 13382)", *Dagstuhl Reports*, 3(9):130, 2014, http://drops.dagstuhl.de/opus/volltexte/2014/4420.

[10] Alex McLean, & Geraint Wiggins, "Tidal - Pattern Language for Live Coding of Music", in proceedings of the 7th Sound and Music Computing Conference (Barcelona: Universitat Pompeu Fabra, 2010), 264.

[11] N. Collins and A. McLean, "Algorave: Live performance of algorithmic electronic dance music", In Proceedings of the International Conference on New Interfaces for Musical Expression (London: Goldsmiths University, 2014), 355.

[12] David Ogborn, "Live coding in a scalable, participatory laptop orchestra", Computer Music Journal, 38(01):17, 2014, https://doi.org/10.1162/COMJ_a_00217.

[13] Tom Hall. "Towards a Slow Code Manifesto", April, 2007, https://ludions.com/texts/2007a/.

[14] Kate Sicchio, "Hacking Choreography: Dance and Live Coding", Computer Music Journal, 38(01):31, 2014, https://doi.org/10.1162/COMJ_a_00218.

What is TidalCycles?

TidalCycles, commonly known as *Tidal* for short, is the creation of Alex McLean. It is a free / open domain specific language (DSL) environment embedded in Haskell[15], for generating algorithmic patterns, that first began around 2009. As described by Alex McLean, "Tidal is designed to allow patterns to be created and modified during a live coded performance, aided by terse, expressive syntax and integration with an emerging time synchronisation standard."[16]

In the past few years, Tidal's popularity grew amongst live coders community, thus making it one of the most used environments for live performance and complex patterns generation. There are certain aspects to its significant growth:

1. Accessibility: The whole Tidal's syntax is pretty much comprehensive to humans, and it has a particular musical taste when writing code. Of course, there is a modest learning curve one has to overcome, but the languages basics are pretty straight forward.

2. Musical Timing: "Underlying this is the assumption that time is structured in terms of rhythmic (or more correctly, metric) *cycles*, a perceptual phenomena that lies at the basis of many great musical traditions including Indian classical (Clayton 2008), and electronic dance music."[17]

3. Ready-made generators and Transformers: Tidal has a great pattern library that contains a broad range of pattern generators and transformers, which can be further manipulated into complex structures. There is also a shared repository with more complex transformers made by Tidal users.

4. Community: Over the past years, a community of Tidal users has started to grow, providing support for its members and openly sharing code, ideas and knowledge.

> "Computers're bringing about a situation that's like the invention of harmony. Sub-routines are like chords. No one would think of keeping a chord to himself. You'd give it to anyone who wanted it. You'd welcome alterations of it. Sub-routines are altered by a single punch. We're getting music made by man himself: not just one man."[18]

At its essence Tidal operates pattering Open Sound Control (OSC) network messages, ingrained in the general-purpose language Haskell. Tidal in mainly used alongside SuperDirt, a hybrid framework for synthesis, MIDI operations and sample manipulation, applied in

---

[15] Alex McLean, "Making Programming Languages to Dance to: Live Coding with Tidal", in proceedings of the 2nd ACM SIGPLAN international workshop on Functional art, music, modeling & design (Gothenburg: Association for Computing Machinery, 2014), 63.

[16] Alex McLean, & Geraint Wiggins, "Tidal - Pattern Language for Live Coding of Music", in proceedings of the 7th Sound and Music Computing Conference (Barcelona: Universitat Pompeu Fabra, 2010), 264.

[17] Alex McLean, "Making Programming Languages to Dance to: Live Coding with Tidal", in proceedings of the 2nd ACM SIGPLAN international workshop on Functional art, music, modeling & design (Gothenburg: Association for Computing Machinery, 2014), 64.

[18] John Cage, *Art and Technology*, Cooper Square Press, 1969.

SuperCollider. Tidal is mainly known as a live coding environment, were musicians write and execute their code in a live performance. It is often used in the context of live coding music, in the dance music scene, in the form of *Algorave*, but it can really be utilized with anything that is pattern-related, such as repetitive dance music, experimental electroacoustic music, minimal music, abstract soundscapes, or any music genre whatsoever. There also has been people using Tidal to make live choreographic scores[19], DMX-controlled lighting, VJing, video art, and even woven textile pattering with algorithms[20]. Tidal's core relays on the concept of algorithmic pattering, so it would be interesting to define what algorithmic pattern actually mean. Grünbaum and Shephard define patterns as "designs repeating some motif in a more or less systematic manner."[21] They are writing their definition in the field of geometric tilings, but the same definition fits perfectly in the context of music.

> "…in music fields, a sequence becomes a pattern, once it is repeated. However, in music we too often focus on the repetition, and not the systematic manner in which it is repeated. For a pattern to be interesting, we need to do more than repeat it; the repetition only provides the metrical ground on which the pattern acts."[22]

In music, any sequence when it gets repeated, is generally called a pattern. But everything is not just about pattern repetitions, for example in textile patterning, there are a lot of patterns that repeat, but there are also patterns that rotate, reflect, turn around or invert themselves, creating symmetry. There are also patterns that play with similarities, but they do it in different scales, patterns that interfere with other patterns. There are also patterns that have glitches making them unpredictable; sometimes these glitches are taken as "errors", but they are often introduced into the patterns on purpose. All these pattering styles can also be explored in TidaCycles. So by the term: "algorithmic pattern", I try to represent the way patterns are perceived, more specifically how the transformations and processes that affect the patterns are perceived in the end result. In summary, an algorithmic pattern is where you see, hear or otherwise perceive ways of making.

Tidal was conceived originally for a live coding situation, despite this it doesn't have to be live coded in front of an audience. There are a lot of artists and musicians, including the author, that prefer to write their lines of code alone in the comfort of their studio, away from the view of the audience and only present the material to the public when they feel ready. Coders would potentially perform it by tweaking, rearranging and manipulating the code they have already written, instead of starting with a blank page and writing the code from zero. Some coders, as the author, would just use these live coding tools as a sonification unit, audio mangler, and as an expressive tool to generate sounds and interesting patterns for their compositions. All the aforementioned options are different possibilities for using Tidal, and none of them are wrong

[19] Kate Sicchio, "Data management part III: An artistic framework for understanding technology without technology", *Media-N: Journal of the New Media Caucus*, 10(01), 2014, https://median.newmediacaucus.org/art-infrastructures-information/data-management-part-iii-an-artistic-framework-for-understanding-technology-without-technology/.
[20] Alex Mclean, "Algorithmic Pattern", in Proceedings of the International Conference on New Interfaces for Musical Expression (Birmingham: Royal Conservatoire, 2020), 265.
[21] Branko Grünbaum and G. C. Shephard, *Tilings and patterns* (New York : W.H. Freeman, 1986), 17.
[22] Alex Mclean, "Algorithmic Pattern", in Proceedings of the International Conference on New Interfaces for Musical Expression (Birmingham: Royal Conservatoire, 2020), 265.

or right. Tidal offers a flexible work environment and is meant to be used as it suits best each coders creative workflow.

Nevertheless, Tidal's core is focused on patterns generation, and it does encourage / influence the user to a certain approach to it. Tidal can be somehow frustrating for users that face it with fixed and rigid ideas, however it will be convenient for users traying to experiment and generate patterns that are tricky for the mind to create. When working with patterns in Tidal, one might have multiple complex layers playing at the same time, resulting in strange interactions that can make it challenging to envision precisely what is going to happen when making any change on one of these particular layers. Anyhow if the users welcome and embrace Tidal's unpredictability, they will get a closer understanding on how its pattern structures operate and grasp a feel for its high-level domain over sequences and its endless capabilities.

Architecture of a Tidal environment

To fully grasp how Tidal operates and have a better idea on what is going behind the scenes when we are typing our code patterns, it is necessary to understand Tidal's under the hood anatomy. So how is Tidal's inner structure designed? TidalCicles fits together as a clustered network of computer programs that interconnect. This cluster is formed by the Tidal library, the Haskell language, the text editor, SuperCollider and SuperDirt.

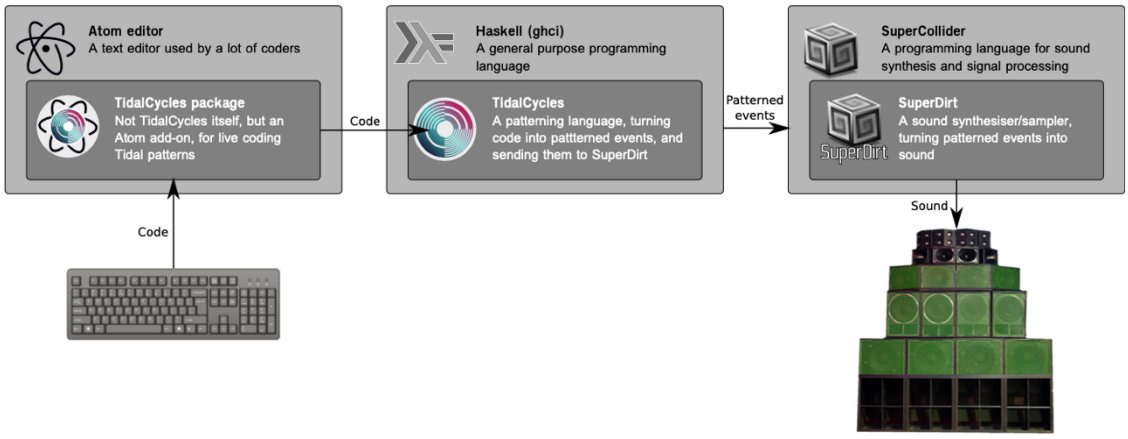

Figure 3 - Configuration of Tidal environment

The diagram displayed above shows the different parts of Tidal's network and how they communicate.
Tidal is written in Haskell, a general purpose programming language and the program that runs Haskell is called "ghci". When we are typing patterns in Tidal, we are actually writing Haskell code, using the Tidal Library as a mediator. Tidal is not just and add-on for the Haskell language, it implements its own computational models and operators for handling patterned structures, so actually it's a programming language in its own right. In computer science terms, it's a domain specific language, embedded in Haskell. Accordingly, there is a structure called "mini-notation" for representing sequences ingrained in tidal. All the pattern generation is made by Tidal; it transforms the code the user inputs into messages, and these are sent to a

sound synthesis program that usually is SuperDirt. Just like Tidal is written in Haskell language, SuperDirt is written in SuperCollider language.

SuperCollider is in a way similar to Tidal, it is a programming environment for synthesis, audio and also a massive digital signal processor (DSP). SuperCollider is one of the most powerful coding environments at the moment of writing this essay, if not the most powerful. We can find SuperCollider operating under the hood of many live coding environments. As a matter of fact, SuperCollider is a great coding environment in its own and there are many live coders that use it as their main coding platform. If you are very knowledgeable and a bit of a super coder, it is possible to create your own synthesizers and effects in SuperCollider, while coding patterns to trigger them in TidalCycles. Just to recap a bit, Tidal is in charge of making the patterns and SuperCollider of making the sound, the one thing that is missing is a space to work with the code, in this case it's a text editor. There are a significant amount of text editors in the market, but few that have plugins for communicating with Tidal; Atom, vscode, emcas or vim, either of this editors will handle Haskell, the loading of Tidal library and making all the connections with SuperCollider. Personally, I use Atom because of its friendly user interface, code display and easy browsing between files, but it doesn't really matter which one to use, they all will just work fine. It is of importance to mention that, as Tidal sends Open Sound Control (OSC) messages, we are not bounded to SuperDirt as the default audio synthesis program. We could use any other music software to produce sound, for example we could use Ableton Live, Pro-tools, Cubase, reaper, or any other DAW that accepts incoming OSC messages. This shows how flexible and open TidalCycles is.

Structure of Tidal patterns

The aim of this section is not to completely understand how to write Tidal code, but to get an idea on how patterns look like and their capabilities. Tidal can be divided in two main parts: the *mini-notation* for easy pattern writing and the *Functions Library* for transforming patterns. The Mini-notation is a malleable way to approaching musical time and also a rapid way to articulating rhythms. Even if you are making 4x4 rolling techno, experimental avantgarde, drone / ambient, polyrhythmic and polymetric music, or any other genre of music, mini-notation will speed your way of writing patterns. Mini-notation is all about pattering / sequencing, it conveys how one event follows after the other, with a repeated and looped structure. At any time there are speech marks (“”) in the code, that will normally mean the presence of a mini-notation sequence. Here we have a straightforward example to better understand the syntax.

```
d1 $ sound "kick snare"
```

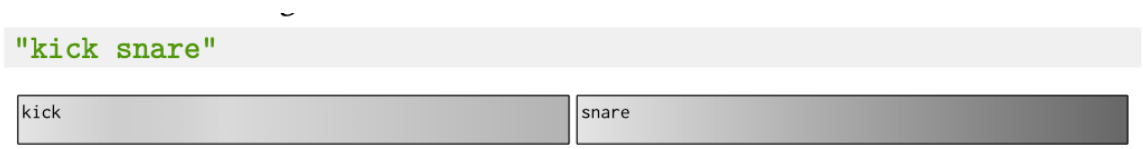

Figure 4 - Simple kick + Snare pattern

The figure 4 shows a simple pattern that plays a snare sound after a kick sound, one after the other in an endless loop. The “kick snare” syntax illustrates the looping mini-notation sequence, the sound statement determines that this is a pattern of sounds, and finally the “d1 $” addresses the pattern that will become sound. Most of the examples I will give in this section of the essay will be visual, rather than musical, so you can have a graphical reference on what is going on with the code.

```
d1 $ sound "orange purple green"
```

```
"orange purple green"
```

Figure 5 - Simple orange + purple + green ternary pattern

The figure 5 also displays a mini-notation pattern, but in this example we can see how adding a new element to the sequence makes all of the events play faster. So Tidal deals with time events in a particular way, not how we are used to understand time in regular digital audio stations or step sequencers.

Cycling-time

TidalCycles has a particular approach to time that we are not used to when dealing with Digital Audio Workstations (DAWS). The majority of these music software, represent musical time, based on the beat. Despite if you are using a software sequencer or just writing your music in sheet, you will commonly express music in relation to a tempo (musical speed) and that tempo will be calculated in beats per minute (BPM). On the contrary, Tidal tends to measure things in cycles rather than in beats, this specific measurement is called CPS: cycles per second. In a musical context, a cycle is equal to a bar or a musical measure. For Tidal, time is cyclical and not linear. It means that when a cycle ends, a new one will follow. Time is counted in smaller and smaller decrements of cycles per second. This rather original way of dealing with time can be quite surprising for a musician, because both traditional European notation and modern sequencers are generally linear and deal with the beginning of time and the ending of time. So how do we deal with cycles? And most important, how are these cycles affecting the patterns?

In the first place, we can observe that the more events we add to a mini-notation pattern, the faster this one will be played. As we can appreciate in the two lines of code displayed in figure 6, the second pattern plays 1.5 times faster than the first one, making all the events of the sequence fit into a single cycle.

```
d1 $ sound "kick snare clap clap"
```


```
d1 $ sound "kick snare clap clap kick kick"
```

Figure 6 - TidalCycles pattern examples

In other music software, such as the forementioned digital audio workstation, we determine the number of beats per bar, and select the tempo in beats per minute (BPM). Yet in tidal, we designate cycles per second, and within a cycle, the whole temporal structure is pattern-fluid. The beats and events can fall all over the place, as well as sequences emerging from complex and compound ratios, creating Polyrhythms and polymeters, time bending patterns and all sort of non-conventional rhythm structures, rather than fixed metrical grid sequences. Thus, the cycle is the main reference point for pattering, and not the event. Which doesn't mean that all cycles must be the same length, or that things must fit inside a cycle.

Graphical Sequencers

Since the sprung of the MIDI protocol, electronic musicians have been exploring the capabilities of sequencers, arpeggiators and keyboards for generating midi patterns. In its simplest expression, sequencers trigger midi events by interpreting an index that includes data such as: pitch, note duration, velocity.

The Sequencer has grown into a universal tool for music composition and may be found in every Digital Audio Workstation (DAW)[23]. Moreover, as Andrew Brown stated, the sequencer developed as a medium for musical expression, a vehicle for communicating music ideas, and a clear and articulate musical voice.[24] In addition to expanding the capabilities of electronic music composers[25].

Graphical sequencers are mediocre tools when it comes to creating complex patterns and odd timing sequences with weird subdivisions, in the judge of this author. By Graphical Sequencers we understand: Step Sequencers, Arppegiators and any commercial software with the capability of sequencing timed events. The user mainly interacts with them by a limited number of keyboard shortcuts and by using the mouse, allowing most of the time insufficient control over the parameters that can be manipulated. Also, as a result of its design the user is indirectly

[23] Francisco Cuadrado, "The use of sequencer tools during the composition process: A field study", *Journal of Music Technology and Education*, 8(01): 55 ,2015, https://doi.org/10.1386/jmte.8.1.55_1.
[24] Andrew R. Brown, Computers in Music Education: Amplifying Musicality (New York: Routledge, 2007), 30.
[25] Paul Théberge, "The End of the World as We Know It: The Changing Role of the Studio in the Internet Age", in The Art of Record Production: An Introductory Reader for a New Academic Field, ed. S. Frith and S. Zagorski-Thomas (Farnham: Ashgate, 2012), 77–90.

influenced during the composition process, and will more likely use on the grid events, 4/4 patterns and semiquaver subdivision rhythms.

The authors earlier experiences with Graphical User Interface (GUI) tools demonstrated to be rigid and inconvenient, when used in live performances, while experimenting with convoluted patterns/sequences and when trying to achieve complex sets of parameter changes simultaneously. This Problem is enlarged taking into account the extensive diversity of Graphical Sequencers that exist. Musicians would like to use them all at the same time but operating them simultaneously is complicated. Graphical Sequencers have separate GUIs that are normally point and click and therefore cannot be simultaneously manipulated with other parameters.

One potential solution to solve this problem could be the usage of programming languages as a tool for live coding music or as a text-based interface sequencer[26]. In a text-based interface, the coder can write out complicated parameter changes and then execute them simultaneously. There are a few computer music languages that can carry out this task, such as: TidalCycles[27], SuperCollider[28], Csound, Chuck[29], IxiLang[30]. Consequently, TidalCycles was specially created for managing these needs. Its Library contains abstractions for musical time manipulation and events sequencing. This allows Tidal to function as a code-based real-time sequencer for any output target that can be connected to the system.

Technology shapes human action

Since the beginnings of electronic music production there is a romantic idealization that new technology is capable of generating any sound imaginable[31]. Even if the previous allegations about music technology are stated as literal or metaphorical, historical experience displays that the range of sounds that early analogue electronics could produce, were in fact extremely limited. Future retrospectives may suggest, that in concern of the control and interaction possibilities, our current DAWs and Graphical Sequencers might be as well narrowly constrained.

As McPherson states, "the design of any tool favours certain types of thinking, certain modes of interaction, certain outcomes over others. It is this non-neutrality in the context of music programming languages."[32] Each individual part of an interface will influence the compositional ideas and mental processes of the musician.

---

[26] Nick Collins, A. McLean, J. Rohrhuber, and A. Ward, "Live coding in laptop performance", *Organised Sound*, 8(03):321, 2003, http://dx.doi.org/10.1017/s135577180300030x.
[27] Alex McLean, "Making Programming Languages to Dance to: Live Coding with Tidal", in proceedings of the 2nd ACM SIGPLAN international workshop on Functional art, music, modeling & design (Gothenburg: Association for Computing Machinery, 2014), 63.
[28] James McCartney, "Rethinking the Computer Music Language: SuperCollider", Computer Music Journal, 26(4): 61, 2002, https://doi.org/10.1162/014892602320991383.
[29] Ge Wang, "On-the-fly Programming: Using Code as an Expressive Musical Instrument", in Proceedings of the International Conference on New Interfaces for Musical Expression (Hamamatsu: Japan, 2004), 138.
[30] Thor Magnusson, "ixiLang: A SuperCollider Parasite for Live Coding", in proceedings of International Computer Music Conference (Huddersfield: UK, 2011), 503.
[31] Paul Théberge, *Any Sound You Can Imagine: Making Music/Consuming Technology* (Connecticut: Wesleyan University Press, 1997).
[32] Andrew P. McPherson and Koray Tahiroglu, "Idiomatic Patterns and Aesthetic Influence in Computer Music Languages", *Organised Sound,* 25(01): 53, 2020, https://doi.org/10.1017/S1355771819000463.

Thor Magnusson characterizes the process of learning a new Digital Musical Instrument (DMI) as "getting a feeling" for the instrument's constrains, rather than engaging with its affordances.[33] Even the most rudimentary Digital Musical Instruments (DMIs) can however arise an extensive range of musical styles[34], and similar studies have stablished that incorporating a higher number of freedom degrees to a DMI, could in the contrary diminish the performer's exploration capacity[35].

Furthermore, constraints on acoustic instruments were usually observed as exciting and inspiring whereas in DMIs constraints were constantly seen as annoying and frustrating. Maybe this feeling could come out of the idea, as Theberge stated, that DMI and new technology should be capable of generating any sound imaginable[36].

On the other hand, a survey carried out by Thor Magnusson and Hurtado Mendieta in 2007 exhibits that imposing constraints in music software was occasionally desirable. Although respondents marked a divergence between graphical interface software that is more musically directive (Ableton Live, Pro Tools, Cubase, Garage Band, Logic Pro, Reaper) and open-ended, text-based music programming software such as Csound, Super Collider, ChucK, Pure Data and Max/MSP[37]. Undeniably, programming languages as the beforementioned, offer higher range of capabilities and minor restrictions in comparison to Digital Audio Workstations, since sounds and sequences can be constructed from its basic foundations.

Essentially, we state that DMI will be shaping the user's actions, by what each music software makes it more obvious, natural and clear to do. Either because of its coding language structure or because its GUIs design.

Live coding performances engages in diverse arrays of decision making and feedbacks than traditional instrumental performances[38], nonetheless there are some procedures that are more organic to a specific language than others. Sergi Jordá introduces into analysis that "music instruments are not only in charge of transmitting human expressiveness like passive channels. They are, with their feedback, responsible for provoking and instigating the performer through their own interfaces."[39]

---

[33] Thor Magnusson, "Designing Constraints: Composing and Performing with Digital Musical Systems", *Computer Music Journal,* 34(4): 65, 2010, https://doi.org/10.1162/COMJ_a_00026.

[34] M. Gurevich, A. Marquez-Borbon and P. Stapleton, "Playing with Constraints: Stylistic Variation with a Simple Electronic Instrument", *Computer Music Journal,* 36(01): 25, 2012, https://doi.org/10.1162/COMJ_a_00103.

[35] V. Zappi, and A. McPherson, "Hackable Instruments: Supporting Appropriation and Modification in Digital Musical Interaction", *Frontiers in ICT,* 5, 2018, https://doi.org/10.3389/fict.2018.00026.

[36] Paul Théberge, *Any Sound You Can Imagine: Making Music/Consuming Technology* (Connecticut: Wesleyan University Press, 1997).

[37] Thor Magnusson and Enrike Hurtado Mendieta, "The Acoustic, the Digital and the Body: A Survey on Musical Instruments", in Proceedings of the 7th International Conference on New Interfaces for Musical Expression (New York: USA ,2007), 96.

[38] Andrew Goldman, "Live Coding Helps to Distinguish between Embodied and Propositional Improvisation", Journal of New Music Research, 48(3): 285, 2019, https://doi.org/10.1080/09298215.2019.1604762.

[39] Sergi Jordà, Digital Lutherie: Crafting Musical Computers for New Musics' Performance and Improvisation, PhD thesis: Universitat Pompeu Fabra, 2005.

In relation to the latter, Thor Magnusson suggests that "instruments are actors: they teach, adapt, explain, direct, suggest, entice. Instruments are impregnated with knowledge expressed as music theory . . . they explain the world"[40]. For example,

> "The piano keyboard 'tells us' that microtonality is of little importance (and much Western music theory has wholly subscribed to that script); the drum-sequencer that 4/4 rhythms and semiquavers are more natural than other types; and the digital audio workstation, through its affordances of copying, pasting and looping, assures us that it is perfectly normal to repeat the same short performance over and over in the same track"[41].

The impact of music programming languages on creative coding practices is not expressed in terms of its limitations, on what is possible to create or not, as programming languages are in theory able to achieve any desirable outcome. Rather, coding languages differentiate in what ideas and constructions are more likely or accessible to code[42]. Therefore, music software shapes musician's actions and more broadly technology shapes human action.

The architecture of any technological tool is ingrained in a social and cultural context, and these tools will embrace the values and ideas embedded in that particular context.[43] Designers, creators and coders of Digital Music Instruments will not only be influenced by their own pre-concepts and aesthetic values[44], but also by the characteristics and inherent attributes of the software they are using to create / code their instruments.

Each computer music language will embrace its own characteristics and particular out-look. As an example, transformation and manipulation of sonic aspects in text-based programming languages will diverge from those that are object oriented.[45] These contrasting operation processes between languages may be more related with differing methods for manipulating data rather than particular aesthetics inherent to each language creators.

The goal of a low-level music language is, as McCarthney explains, to "provide a set of abstractions that makes expressing compositional and signal processing ideas as easy and direct as possible. The kinds of ideas one wishes to express, however, can be quite different and lead to very different tools"[46].

With TidalCycles we can operate from low-level unit generators, over to controlling SynthDefs ("blueprints for synthesis units"), to complex pattern explorations or audio manipulation

---

[40] Thor Magnusson, "Ergomimesis: Towards a Language Describing Instrumental Transductions", in proceedings of the International Conference on Live Interfaces (Porto: Portugal, 2018), 79.
[41] Thor Magnusson, "Of Epistemic Tools: Musical Instruments as Cognitive Extensions", *Organised Sound,* 14(2):171, 2009, https://doi.org/10.1017/S1355771809000272.
[42] Chris Nash, "The Cognitive Dimensions of Music Notations", In proceedings of the International Conference on Technologies for Notation and Representation (Paris: IRCAM, 2015), 191.
[43] Lucy A. Suchman, *Plans and Situated Actions: The Problem of Human-Machine Communication* (Cambridge: Cambridge University Press, 1987).
[44] G. Lepri and A. McPherson, "Making Up Instruments: Design Fiction for Value Discovery in Communities of Musical Practice", in proceedings of the ACM Conference on Designing Interactive Systems (San Diego: USA, 2019), 113.
[45] Eric Lyon, "Dartmouth Symposium on the Future of Computer Music Software: A Panel Discussion", *Computer Music Journal,* 26(4): 13, 2002, https://doi.org/10.1162/014892602320991347.
[46] James McCartney, "Rethinking the Computer Music Language: SuperCollider", *Computer Music Journal,* 26(4):1, 2002, https://doi.org/10.1162/014892602320991383.

through digital signal processing chains. The foremost important features here are the combination of flexibility (the ability of live coding, rearranging the code on the fly) and the possibility of creating abstractions (such as custom variables, SynthDefs and classes).

As Ihde states, technology influences people`s behavior and their surroundings without being consciously perceived. The hypothesis that technology is a non-neutral tool that shapes human cognition (actions and processes) is deep-rooted in philosophy.[47] An illustration to this influence can be observed in that live coding is not an abstract practice, but instead one that experiments and answers to the affordances and constraints of the programming tools that are being used. When utilizing music programming tools, decision making is usually made "in situ", in reaction to the immediate events and opportunities that the surroundings and the tools offer.

If all the music software had the same capabilities and favored the same working procedures, we could probably ask ourselves why there is such a huge variety of music programs and many of them are backed by active user communities. Every music software will definitely encourage specific ideas and pattern creations, depending on what's more accessible or suitable in that environment. In the same way that traditional instrument constrains shape improvisation and composition, similar influences occur with music programming languages, which affect the in-situ coding exploration and pattern creation.

The Interface

As all musicians, coders and producer who have experience working with digital software know, the software silently influences your workflow, suggesting an inherent methodology to be adopted. As Thor Magnusson explains, "The interface is an instrument. It is a graphical manifestation of musical ideas and work processes. An interface is at the same time the aesthetic platform defining musical structures and the practical control-base for the underlying soundengine."[48] Furthermore, he states on the same subject, "The interface … is an important factor in a musical performance. It can evoke emotions, encourage direct responses, trigger ideas and open up unknown paths in a live performance."[49] In addition, he explains that "The creation of an interface is essentially the creation of a semiotic system that affects and influences the musician and the composer."[50]

The interface constructs possibilities (affordances), but also defines limitations (constrains) of what can be achieved with any software. Here we are principally talking about Graphical User Interfaces of Audio Software, but this concept can be further expanded to audio programming languages as well. The type of language structure or ready-made objects determinates what can be expressed, for Example Digital Audio Workstations (DAWS) start with a predetermined graphical interface, with pre-fabricated ready-made tools for general music purposes. Most Graphical Sequencers use knobs, faders and buttons, they work in a linear sequencing way (like step sequencers and arpeggiators), using a keyboard role and fixed-pitch (tempered scale and discrete pitch organization), and they usually follow a musical timeline (Bpm / tempo oriented).

---

[47] Don Ihde, *Technology and the Lifeworld: From Garden to Earth* (Indianapolis: Indiana University Press, 1990), 24.
[48] Thor Magnusson, "ixi software: The Interface as Instrument", in Proceedings of NIME (Vancouver: University of British Columbia, 2005), 212.
[49] Ibid.
[50] Thor Magnusson, "Screen-Based Musical Interfaces as Semiotic Machines**",** in Proceedings of the International Conference on New Interfaces for Musical Expression (Paris: France, 2006), 162.

On the other hand, in Music Programing environments there is no user interface, the environments start up with a blank page a tabula rasa, allowing them to create from scratch every sound imaginable.

Musical software exists in various configurations and in this essay, I separate them into two main formats: music productions and programming language tools. The main contrast between a music production tool and a text-based programing language, is that the former has an intrinsic interaction design, that has a much stronger aesthetic influence on the user than the latter. Software will never be an expressive neutral tool, the more polished and refined it is, the more it will influence the user.

When Musicians incorporate music software in their compositions, their workflow is somehow dictated by the software, they must shape their working process to adapt the interface and architecture of the software. To a same degree as with acoustical instruments, software determinates the capacity of expression. The Musician is already immersed in a cluster of its own pre-concepts and methodic thinking, however the degree of flexibility and expressive freedom rests on the environment they are operating. From this viewpoint TidalCycles, and other programing languages as: Super Collider or Pure Data, are indeed more free and open-ended than Ableton Live, Logic Pro, Protools or any other DAW. Thus, the musical cognitive process occurs within the components that build the interface of the software.

So as to prevent the suggestive influence Graphical Sequencers have on musicians, these are gradually shifting towards free and open-source programming environments. The concept of "free" is used here implying the liberation of musical expression dictated and ingrained in the interface design of these tools, as a distinctive quality of their nature. Opposed to Graphical Sequencers, there are as many different ways of using Tidal as there are people using the software. There is no fixed method or procedure to approaching programming environments, as all programmers / Musicians have their personal mindset and workflow when writing code / Music. Therefore, TidalCycles is a flexible environment that has the capability of running wide number of patterns and parameter changes at the same time. All the operations can be mangled, re-worked, restructured, without interrupting the rest of the processes.

Conclusion

In this essay I showed how Graphical sequencer are rigid and inconvenient tools for experimenting with convoluted patterns / sequences and when trying to achieve complex sets of parameter changes simultaneously. A potential solution offered for solving this problem was the use of programming language environments and more specifically text-based programming environments, thus they allow the musician to create more complicated patterns and parameter changes, while executing them simultaneously. For this case, the structure and functioning of TidalCycles was explored and analysed.

I also shined light in the fact that technology is a non-neutral tool, transforming music software into an instrument that shapes musician's cognition. As I previously mentioned in this essay, Graphical Sequencers, with their intrinsic interaction design, have a much stronger aesthetic influence over musicians than text-based programming environments.

I also discussed the problem with software interfaces, it is a graphical manifestation of musical ideas and work processes. I arrived at the conclusion that regarding its interface layout, text-based programming environments enable a wider space for experimentation, simultaneous parameters control, complex sequence creation and sophisticated audio manipulation than graphical sequencers. And a way to drift away from graphical sequencers interface influences, is to swich to a text-based programming environment, consequently the one suggested in this essay is TidalCycles.

There is no fixed method or procedure to approaching text-based programming environments, as every Musician has its own reasoning and workflow when writing Music. Therefore, TidalCycles is a flexible environment, that has the capability of running wide number of patterns and parameter changes at the same time. All the operations can be mangled, re-worked, restructured, without interrupting with the rest of the processes.

Bibliography

Blackwell, A. McLean, A. Noble, J. and Rohrhuber ,J. “Collaboration and learning through live coding (Dagstuhl Seminar 13382)”. *Dagstuhl Reports*, 3(9):130–168, 2014. http://drops.dagstuhl.de/opus/volltexte/2014/4420.

Brown, Andrew R. *Computers in Music Education: Amplifying Musicality.* New York: Routledge, 2007.

Cage, J. *Art and Technology*. Cooper Square Press, 1969.

Collins, N. McLean, A. Rohrhuber, J. and Ward, A. “Live coding in laptop performance”. *Organised Sound*, 8(03):321–330, 2003. http://dx.doi.org/10.1017/s135577180300030x.

Collins, N. and McLean, A. “Algorave: Live performance of algorithmic electronic dance music”. In Proceedings of the International Conference on New Interfaces for Musical Expression, 355-358. London: Goldsmiths University, 2014.

Cuadrado, Francisco. “The use of sequencer tools during the composition process: A field study”. *Journal of Music Technology and Education*, 8(01): 55-70 ,2015. https://doi.org/10.1386/jmte.8.1.55_1.

Goldman, Andrew. “Live Coding Helps to Distinguish between Embodied and Propositional Improvisation”. *Journal of New Music Research,* 48(3): 281–293, 2019. https://doi.org/10.1080/09298215.2019.1604762.

Grünbaum, Branko and Shephard, G. C. *Tilings and patterns.* New York: W.H. Freeman, 1986.

Gurevich, M. Marquez-Borbon, A. and Stapleton, P. “Playing with Constraints: Stylistic Variation with a Simple Electronic Instrument”. *Computer Music Journal,* 36(01): 23–41, 2012. https://doi.org/10.1162/COMJ_a_00103.

Hall, Tom. “Towards a Slow Code Manifesto”. April, 2007. https://ludions.com/texts/2007a/.

Holms, Thom. “Early Computer Music (1953-85)”. In *Electronic and Experimental Music: Technology, Music and Culture,* 291-313. New York: Routledge, 2016.

Hugill, Andrew. “Performance and Musicianship: Live Coding”. in *The Digital Musician Second Edition,159-179.* New York: Routledge, 2012.

Ihde, Don. *Technology and the Lifeworld: From Garden to Earth*. Indianapolis: Indiana University Press, 1990.

Jordà, Sergi. Digital Lutherie: Crafting Musical Computers for New Musics’ Performance and Improvisation. PhD thesis: Universitat Pompeu Fabra, 2005.

Lepri, G. and McPherson, A. “Making Up Instruments: Design Fiction for Value Discovery in Communities of Musical Practice”. In proceedings of the ACM Conference on Designing Interactive Systems, 113- 126. San Diego: USA, 2019.

Lyon, Eric. “Dartmouth Symposium on the Future of Computer Music Software: A Panel Discussion”. *Computer Music Journal,* 26(4): 13–30, 2002. https://doi.org/10.1162/014892602320991347.

Magnusson, Thor. “Confessions of a Live Coder”. in Proceedings of International Computer Music Conference, 1-8. Huddersfield: University of Huddersfield, 2011.

Magnusson, Thor. “Designing Constraints: Composing and Performing with Digital Musical Systems”. *Computer Music Journal,* 34(4): 62–73, 2010. https://doi.org/10.1162/COMJ_a_00026.

Magnusson, Thor. “Ergomimesis: Towards a Language Describing Instrumental Transductions”. In proceedings of the International Conference on Live Interfaces, 78-85. Porto: Portugal, 2018.

Magnusson, Thor. “ixiLang: A SuperCollider Parasite for Live Coding”. In proceedings of International Computer Music Conference, 503-506. Huddersfield: UK, 2011.

Magnusson, Thor. "ixi software: The Interface as Instrument". in Proceedings of NIME, 212-215. Vancouver: University of British Columbia, 2005.

Magnusson, Thor. “Of Epistemic Tools: Musical Instruments as Cognitive Extensions”. *Organised Sound,* 14(2):168-176, 2009. https://doi.org/10.1017/S1355771809000272.

Magnusson, Thor. “Screen-Based Musical Interfaces as Semiotic Machines**”.** In Proceedings of the International Conference on New Interfaces for Musical Expression, 162-167. Paris: France, 2006.

Magnusson, Thor and Hurtado Mendieta, Enrike. “The Acoustic, the Digital and the Body: A Survey on Musical Instruments”. In Proceedings of the 7th International Conference on New Interfaces for Musical Expression, 94-99. New York: USA ,2007.

McCartney, James. “Rethinking the Computer Music Language: SuperCollider”. *Computer Music Journal,* 26(4): 61–68, 2002. https://doi.org/10.1162/014892602320991383.

Mclean, Alex. “Algorithmic Pattern”. in Proceedings of the International Conference on New Interfaces for Musical Expression, 265–270. Birmingham: Royal Conservatoire, 2020.

McLean, Alex. “Making Programming Languages to Dance to: Live Coding with Tidal”. in proceedings of the 2nd ACM SIGPLAN international workshop on Functional art, music, modeling & design, 63-70. Gothenburg: Association for Computing Machinery, 2014.

McLean, A. Rohrhuber, J. and Collins. N. “Special issue on Live Coding: Editor’s notes”. *Computer Music Journal*, 38(01):4-5, 2014. https://doi.org/10.1162/COMJ_e_00225.

McLean, Alex, & Wiggins, Geraint. “Tidal - Pattern Language for Live Coding of Music”. In proceedings of the 7th Sound and Music Computing Conference, 264-269. Barcelona: Universitat Pompeu Fabra, 2010.

McPherson, Andrew P. and Tahiroglu, Koray. “Idiomatic Patterns and Aesthetic Influence in Computer Music Languages”. *Organised Sound,* 25(01): 53 – 63, 2020. https://doi.org/10.1017/S1355771819000463.

Nash, Chris. “The Cognitive Dimensions of Music Notations”. In proceedings of the International Conference on Technologies for Notation and Representation, 191-203. Paris: IRCAM, 2015.

Ogborn, David. “Live coding in a scalable, participatory laptop orchestra”. *Computer Music Journal*, 38(01):17-30, 2014. https://doi.org/10.1162/COMJ_a_00217.

Puckette, Miller. “The Patcher”. in Proceedings of International Computer Music Conference, 420-429. San Fransico: ICMA, 1988.

Sicchio, Kate. “Data management part III: An artistic framework for understanding technology without technology”. *Media-N: Journal of the New Media Caucus*, 10(01), 2014. https://median.newmediacaucus.org/art-infrastructures-information/data-management-part-iii-an-artistic-framework-for-understanding-technology-without-technology/.

Sicchio, Kate. “Hacking Choreography: Dance and Live Coding”. *Computer Music Journal*, 38(01):31–39, 2014. https://doi.org/10.1162/COMJ_a_00218.

Suchman, Lucy A. *Plans and Situated Actions: The Problem of Human-Machine Communication*. Cambridge: Cambridge University Press, 1987.

Théberge, Paul. *Any Sound You Can Imagine: Making Music/Consuming Technology.* Connecticut: Wesleyan University Press, 1997.

Théberge, Paul. “The End of the World as We Know It: The Changing Role of the Studio in the Internet Age”. In The Art of Record Production: An Introductory Reader for a New Academic Field, ed. S. Frith and S. Zagorski-Thomas, 77–90. Farnham: Ashgate, 2012.

Wang, Ge. “On-the-fly Programming: Using Code as an Expressive Musical Instrument”. In Proceedings of the International Conference on New Interfaces for Musical Expression, 138–143. Hamamatsu: Japan, 2004.

Zappi, V. and McPherson, A. “Hackable Instruments: Supporting Appropriation and Modification in Digital Musical Interaction”. *Frontiers in ICT,* 5, 2018. https://doi.org/10.3389/fict.2018.00026.