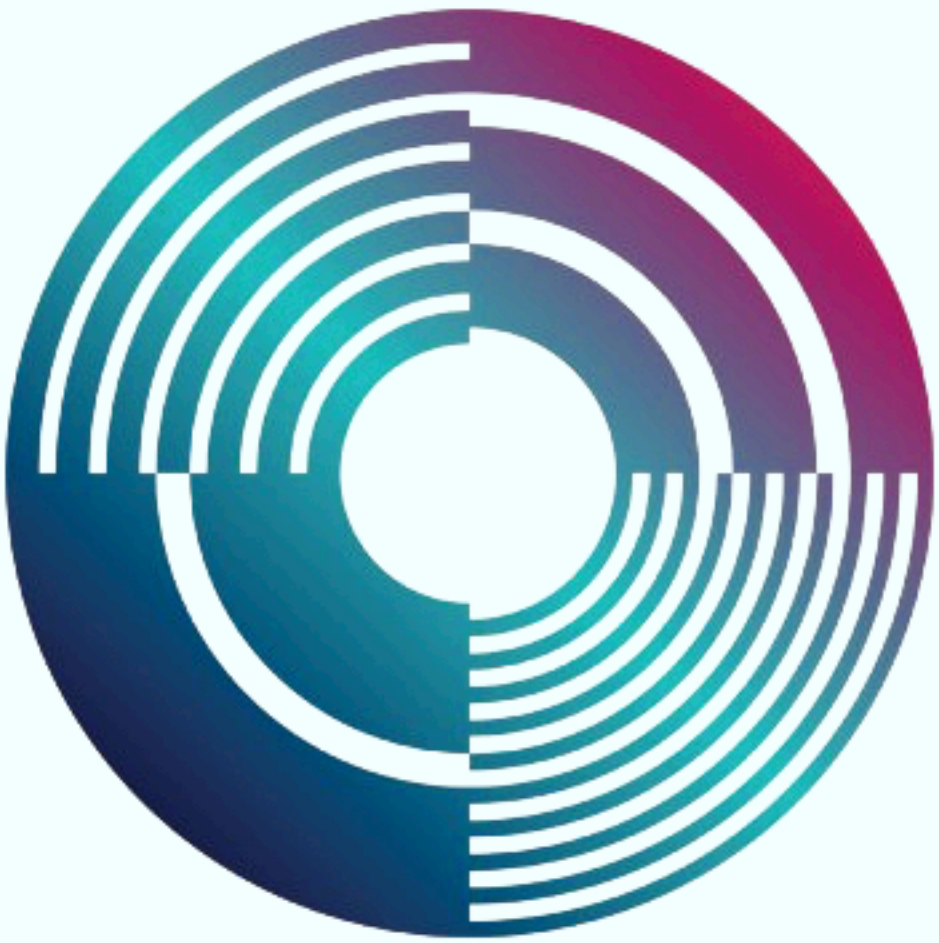
USING A LIVE CODING AS A MEDIUM FOR COMPOSITION:

TIDALCYCLES

Candidate Number: 196484
Module Name: Composition
Module Code: 804W3
Academic Year: Autumn 2018



ABSTRACT

In this paper, I explore the practice of live coding within composing computer music. I outline existing research into the use of live coding languages such as SuperCollider and TidalCycles, and their advantages and disadvantages within the context of composition, as well as themes of authorship in the generative arts. I present contemporary examples of live coding systems and performances, and describe the processes used in my composition birds.tidal. Finally, I critically analyse my piece and the methods I used to learn live coding - of which I have no prior experience within the field of composition or performance.

Keywords: computer music, live coding, live composition, improvisation

INTRODUCTION

Alex McLean, researcher, and creator of TidalCycles defines live coding as the writing of rules in real-time, in order to “improvise time based art such as music, video animation or dance” (McLean and Wiggins, 2010). Well known in the live-coding scene, McLean is mostly known for his and Nick Collins’ (previous Sussex lecturer and now Durham university reader) coining of the term “Algorave” - a genre of music that is composed and performed through the use of coding languages rather than pre-made software packages such as digital audio workstations (Algorave, 2019). In Algorave there is a definite importance in the interaction between the audience and the live-coder as with most performative media, however for this brief, I decided to engage with live-coding on a compositional basis, moreover, I was interested in investigating what the medium could offer me as a composer with regards to accessing complex rhythms and time-based effects.

SuperCollider is one of the most renowned open-source platforms for audio synthesis and algorithmic composition. It being open-source has allowed third party developers such as McLean to create software that uses SuperCollider’s synthesis engine, but uses a different syntax or language, which in-turn affords users a higher-level interaction with it. ‘Higher-level’ here means possessing a greater amount of abstracted detail from the inner workings of the software - or machine language, or if I were to give you a higher-level definition of it, ‘a simpler interaction’! This generally results in a less verbose language, and one of the primary benefits of this is the ability to code quicker and with a reduced probability of error. I chose TidalCycles because of this, as well as the fact that it has an active user-base with forums that explain clearly the functions of the language (Tidalcycles.org, 2019). I have worked with SuperCollider in the past at a very basic level and so I was confident with setting up the server for Tidal (which it needs before each session). Tidal works on a looping structure, where samples are played on a per cycle basis. Most language functions and processes make use of the cycle speed to inform their effect.

RESEARCH BASIS

Live coding in the context of computer music, is a medium in which music is generated in real time by sets of rules or algorithms defined by the composer. Its use in Algorave culture has seen the word live to wrongly attributed to the medium being solely a performative one, whereas 'live' actually refers to the real-time nature of the code evaluation (McLean and Wiggins, 2010). Live coding as a medium for creative expression is one that has seen increased use across many types of genres, not just Algorave. In this section I will explore contemporary literature surrounding the medium, and examples of its use across different genres and creative practices.

The members of the live-coding duo "aa-cell" describe processes in live coding as assuming the following five criteria (Brown and Sorensen, 2009):

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

Brown and Sorensen write about processes as being one of the most important elements of live coding. As an example, below is a function (TidalCycles nomenclature for process) of which the effect is shifting a sample backwards 0.25 seconds every 2, 3, then 4, seconds.

```
$ foldEvery [2,3,4] (0.25<~)
```

Figure 1

Having this level control over mathematical patterns allows composers to make complex systems in real time due to the relatively short language elements. The fact that everything is portrayed in integer values means you can simply perform mathematical functions to change effects and processes. However, as described by Brown and Sorensen, it means that scores for live coded music aren't created - because algorithms are changed in real time, similar to other live improvisational media.

Collins et al. critically analyse the benefits of using live coding as a creative practice in their paper “Live coding in laptop performance” (Collins et al., 2003).

> Although there is great flexibility in afforded to users of live coding languages, sometimes it can be easy to get lost in your own code, and it can take too long to prepare the next section of music.
>
> It presents a new form of improvisation, however it is one that is relatively merciless - there are some “ugly moments of sound” if you mistype a line of code.
>
> On the one hand, it is rewarding to see real efforts translated into sound. On the other hand, coding on the fly without previously tested “snippets” of code can end in dead ends in cases where inspiration is not forthcoming.
>
> Computer languages afford users immensely rich and infinite grammars, but typing isn’t always the most inspirationally stimulating method of interfacing with creative work.

It would be very hard for the process in Figure 1 to be thought of in real-time by an improvisational musician not using live coding language, an aspect that provides a lot of appeal to the medium. Critically, however, this has brought into question the authorship of the music created by live coding processes. Generative artist, software designer and co-musician of ‘Slub’ with Alex McLean, Adrian Ward, explains that Roland Barthes’ post-structuralist “Death of the Author” theory (Barthes, 1994) does not apply to these processes, by re-iterating the importance of the author’s own creativity in the creation and implementation of these algorithms and generative processes (Ward and Cox, 2004).

Some inspiring examples of recent live coding performances and tools that I have come across in my research into this creative practice include the experimental rock band 65daysofstatic’s use of McLean’s TidalCycles in their latest studio session. Here, they use the language to generatively trigger samples and hardware synthesisers as a way of freeing themselves to play other instruments. The musician controlling the TidalCycles patch as pivotal as the rest of the musicians within the piece (Sony, 2018). Thor

Magnusson's Threnoscope aims to visually and descriptively score live coding drone compositions by mapping them to morphing sections of rotating circles rotating around a common origin. It "encourages a certain way of thinking when composing, where tunings, harmonics, harmonic beating, scales, and slow movements are emphasised" (Magnusson, 2013). I found Mike Hodnick's a.k.a Kindohm live set in TidalCycles at the International Conference on Live Coding extremely expressive and motivational (Hodnick, 2016).

BIRDS.TIDAL

To quickly demonstrate the main reason I used TidalCycles over SuperCollider for this brief, here is an example of the difference in verbosity between the two languages when coding a kick drum to play once per second:

TidalCycles

```
setcps 1          Set cycles per second amount (tempo)
d1 $ s "bd"       Play kick drum sample from library per cycle
```

SuperCollider

```
b = Buffer.read(s,"/downloaded-quarks/Dirt-Samples/bd/BT0A0A7.wav");   Commit sample to buffer
x = {PlayBuf.ar(1, b, trigger: Impulse.kr(~bpm/60))}.play;            Play buffer at tempo
```

I'm not trying to frame one language as being 'better' than the other, but coding with TidalCycles is definitely quicker. One of the main downsides to this, ironically, is that it makes assumptions. However, these assumptions (panning set to centre, gain set to full, etc.) can be changed as quickly, and before you run the instrument code, as shown in birds.tidal.

I set out to learn TidalCycles without any live coding experience, by learning first via the online reference of the language's functions. I quickly learnt of the power of the language through applying its time based effects and processes. Two of the earliest functions I learnt were [$ slow] and [$ striate]. [$ slow] takes one argument (input integer in this case) i.e. [$ slow 4]. This slows down the sample it is applied to by 4 times. [$ striate] splits up the applied sample into a specified amount of grains. When used together, it makes an interesting and rich effect, especially when you use it in conjunction with the [$ note] function which when used like so [$ note "1*16"], plays the sample 16 times per cycle. In

my composition “birds. tidal” this can be heard at around the 10 minute mark, and as soon as [$ note] is used to multiply the bird song sample more than 1-2 times per cycle, the original sample starts to become unintelligible melodically (original sample plays at 6:30, use of [$ note] starts at 7:38). As can be heard, the once melodic sample of birdsong becomes one of a more rhythmic nature.

One of the more interesting rhythmic effects can be seen in instrument 2,”d2”, which spends the majority of its time being the hi-hat and snare sample. The piece starts off with a simple crochet beat on the hi-hat and a minim beat on the snare (2:52). After a few repetitions, I use the, [$ whenmod], and [$ stut] functions together to increase the rhythmic complexity. When used together, the function has the following effect, highlighted in red are the parameters of the functions:

d1 $ whenmod 8 4 (stut 8 (1/3) (28/4)) $ s “[hh*4,[~ sd]]”

$ whenmod x y (z): “divide current cycle number by x, if remainder is greater than y, do x”

$ stut x y z: “repeat sample x times, each one y times the volume of the last, spread out over z cycles”

Using the values that I did resulted in emergent rhythms, in that they were ones that I didn’t fully expect, especially when in a live context. I believe this is all part the medium - improvising with different values, learning what you can and can’t do with certain functions. For example, once I [$ striate]’d too hard, it crashed my computer with a kernel panic error! A harsh lesson in learning the upper bounds of that function, I thought!

One of the module readings I engaged with the most was computer scientist and interdisciplinary researcher Godfried Toussaint’s 2005 paper on the Euclidean algorithm generating traditional music rhythms (Toussaint 2005). In this paper, Toussaint outlines how the greatest common divisor of two numbers generates most traditional world music rhythms. It was just my luck that this is a function that McLean has added to TidalCycles. It can be heard in the bass synth sound (4:58), the rhythm of which is [3,8] in terms of its Euclidean algorithm - and is a traditional West African rhythm.

DISCUSSION / CONCLUSION

I learnt TidalCycles over the course of several weeks. The first two weeks were spent primarily learning and experimenting with the language's functions. After that, I only ever engaged with it in the context of live composition - recording myself and streaming myself to popular live streaming video platform Twitch.tv. When in the context of live composition, while retrospectively I'm pretty certain I had no one viewing my stream, there was clear pressure to make decisions on (and in) time, and decisions that would lead to consonant and interesting music for whomever listened to my music. However, due to the nature of live improvisation and composition, there are clear errors in birds.tidal. There are moments when the gain is too loud because I forget to set it under time pressures, moments where I don't get the pitch quite right because I'm used to dealing with pitch in the arbitrary yet popular chromatic scale, not as integers. These are all problems that I'm sure would be ironed with more practice, perhaps by employing exercises as advocated by Collins (Nilson, 2007)

I believe birds.tidal shows clearly the benefits of using live coding as a tool for composition. Although the structure isn't as complex as it could be due to time constraints of live coding in a language I don't have too much experience in, what it has taught me is that TidalCycles is definitely a coding language that allows fast real-time effects processing that leads to making interesting and rhythmically complex patterns, and one that I would like to learn more of. Using computer languages to code music in real-time, while challenging for several reasons, is definitely possible.

BIBLIOGRAPHY

Algorave. (2019). *About Algorave*. [online] Available at: https://algorave.com/about/.

Barthes, R., 1994. 11 The Death of the Author. *Media Texts, Authors and Readers: A Reader*, p.166.

Brown, A.R. and Sorensen, A., 2009. Interacting with generative music through live coding. *Contemporary Music Review*, *28*(1), pp.17-29.

Collins, N., McLean, A., Rohrhuber, J. and Ward, A., 2003. Live coding in laptop performance. *Organised sound*, *8*(3), pp.321-330.

Hodnick, M. (2016). *Kindohm Live @ ICLC 2016, Hamilton, Ontario*. [video] Available at: https://youtu.be/smQOiFt8e4Q.

Magnusson, T., 2013. The Threnoscope: A musical work for live coding performance. In *ICSE*. Live 2013.

McLean, A. and Wiggins, G.A., 2010. Live Coding Towards Computational Creativity. In *ICCC* (pp. 175-179).

Nilson, C., 2007. Live coding practice. In Proceedings of the 7th international conference on New interfaces for musical expression (pp. 112-117). ACM.

Sony (2018). *65daysofstatic Studios Session*. [video] Available at: https://www.youtube.com/watch?v=_OAKUlRlNF0.

Tidalcycles.org. (2019). *TidalCycles userbase*. [online] Available at: https://tidalcycles.org/index.php/Welcome.

Toussaint, G. 2005. The Euclidean algorithm generates traditional musical rhythms. In *Proceedings of BRIDGES: Mathematical Connections in Art, Music and Science* (pp. 47-56).

Ward, A. and Cox, G., 2004. The Authorship of Generative Art. In *Proceedings of Generative Art*.