Scoring with Code: Composing with algorithmic notation

THOR MAGNUSSON

Department of Music, School of Media, Film and Music, Silverstone 214, University of Sussex, Brighton, BN1 9RH, UK
Email: t.magnusson@sussex.ac.uk

Computer code is a form of notational language. It prescribes actions to be carried out by the computer, often by systems called interpreters. When code is used to write music, we are therefore operating with programming language as a relatively new form of musical notation. Music is a time-based art form and the traditional musical score is a linear chronograph with instructions for an interpreter. Here code and traditional notation are somewhat at odds, since code is written as text, without any representational timeline. This can pose problems, for example for a composer who is working on a section in the middle of a long piece, but has to repeatedly run the code from the beginning or make temporary arrangements to solve this difficulty in the compositional process. In short: code does not come with a timeline but is rather the material used for building timelines. This article explores the context of creating linear 'code scores' in the area of musical notation. It presents the *Threnoscope* as an example of a system that implements both representational notation and a prescriptive code score.

1. INTRODUCTION

Live coding has often been portrayed as a practice that merges real-time composition and performance in a participatory context (Collins, McLean, Rohrhuber and Ward 2003; Sorensen and Brown 2007; McLean 2008). Audience members follow the activities of performers writing scores in the form of code to be executed by the computer language interpreter. As in traditional notation, the practice involves the notation of musical intentions through a set of encoded rules that are understood and performed by an interpreter. The principal difference is that in live coding music is typically composed in real time by means of writing algorithms. Whilst algorithms can be written for human interpreters, they are indigenous elements of live coding languages, where performers programme events into the future to be executed by a system slaved to an underlying tempo-clock. Through repeated practice and performance, the live coder builds up techniques of mentally carrying representations of abstract machines operating in time, in the form of loops or temporally recurrent functions that have been given identifying names, and which can be altered during the performance.

Conceived of as a musical score, code is not a direct one-to-one mapping between signs and sound; between signifiers and the signified. Since algorithms are often non-deterministic – with parameters affecting performance deriving from the environment, artificial intelligence or random operations – code reveals a space of possibilities, not unlike the open scores composed by Cage, Wolff, Pousseur or Feldman in the middle of the twentieth century. The written notation can be simple – a few lines of code calling other functions – but the composition might equally result in series of infinite variations and patterns of perfect repeatability. Complicating the idea of seeing code as a direct and ostensibly causal representation for the music it generates is the fact that functions can write other functions, where code composes itself, and nothing is written or materialised that represents the sound, only temporary structures in the interpreter's memory. For a non-programmer, it might therefore be hard to imagine how the static text results in continuous musical events that evolve in time. Any attempt at creating graphic notations of code's functionality as it will be executed in time is practically impossible: the textual language is itself its best notation and its output is the ideal representation (there is no need for a level of abstraction here; cf. the Borgesian map).[1]

Musical notation has typically followed the tradition of defining events on a linear timeline. This representation of time on a traditional stave is not perfectly isomorphic, as bars with notes of long duration are typically shorter on the paper than bars with many short notes. Such compression and expansion of 'spacetime' shows that the language of the score is event-based as opposed to time-based, a fact that might partly originate from concerns of saving paper. However, it should be noted that chronographic timelines of any kind, such as historical events where

[1]There are some interesting projects coming up that visualise the functionality of code through tracing its operations. It is important to stress that these are not representations of code or its output, but of its functionality, allowing the programmer to observe the internal workings of the code. TraceGL (https://trace.gl) is a good example of such a project. That said, there is much to be explored in the development of secondary notation of code (see, for example, the work of Thomas R. G. Green and Marian Petre (1996)), where graphic symbols can add to the already impressive array of useful techniques.

Organised Sound **19**(3): 268–275 © Cambridge University Press, 2014. doi:10.1017/S1355771814000259

time is mapped directly to space, were uncommon in medieval times when the foundation of our current musical notation was established, and only begin to appear in the mid-eighteenth century, for example with Barbeu Dubourg's *Machine Chronographique* (Boyd Davis 2012). In the field of music the piano-rolls in the nineteenth century were the first time-based, arithmetically extended timelines for musical events. This form of notation has been continued in the design of digital music workstation timelines, although printed musical scores do still tend to be event-based.

The linear timelines of staff notation or digital audio workstations seem to be slightly incongruous with code, as the former encourages deterministic and fixed approach to composition, whereas the latter opens up for indeterminacy, non-linearity and liveness. However, it is important that this dichotomy between the linearity of the score and the generativity of code, as presented here, is not understood as two separate and incompatible traditions: there exists a tradition of representing code on linear timelines or what this article defines as 'code scores'. Furthermore, traditional staff notation does share some algorithmic features with code, with its use of elements such as repeat signs, da capo and coda, greatly extended by the diverse bespoke notational symbols, textual instructions and graphic notation invented by individual composers or specific musical traditions.

This article will explore both historical issues and implementation factors related to scoring code on timelines. The context is live coding practice, and a concrete example is presented, the *Threnoscope* – a system for microtonal live coding performance, implementing both a representational notation of the system's state and a graphical prescriptive score language for code.

2. REPRESENTATIONAL NOTATION

Musical notation is a systematic format of instructions for the performance of musical events. It is a prescriptive code allowing agents who read it, whether humans or machines, to perform the music to a varied degree of sophistication, depending upon practice, skill and understanding of the format. As a coherent code, musical notation is a system of abstractions, affording certain types of expression but excluding other. Any system of abstractions has involved a design process of deciding what to include and exclude, judged from a perspective of what is important to express (Bowker and Star 2000; Seeger 1958). It is generally accepted that the human performer will add considerably to the music, or 'fill it with life' through the process of interpretation, as if impregnating the music with the life that got extinguished when it was written down in notational form. We thus find in Western musical notation a reasonably good support for notating twelve-tone equal tempered pitch, metrical rhythm and duration, but a less sophisticated language for dynamics, timbre and microtonal composition. This system of notation strongly references keyboard instruments, where pitch is discrete, as opposed to, say, the violin or the trombone. There are symbols denoting continuous changes in pitch, dynamics and timbre, but those are secondary notations, often of idiosyncratic nature. In the case of computer notation there is an array of formats, from the simple MIDI protocol that has been criticised for its insufficient syntax and resolution, to more open and flexible protocols such as MusicXML (Good 2001), or OSC (Wright 2005), where practically any musical parameter can be specified to the desired resolution. An interesting feature of OSC is the lack of musicality in the protocol; it is 'open' to include any possible parameter or feature description of the music.

Musical notation has often been named as one of the key features contributing to the complexity of Western music. Notation is a mnemonic device offering composers the possibility of externalising their thoughts and operate with blocks of musical text as scaffolding that frees them from having to keep all the parameters in mind at once (Clark and Chalmers 1998). Through the use of graphic symbols, the composer is able to think with external building blocks that can support the most consummate composition of counterpoint, chord progressions or polyphonic music. The ethnomusicologist Bruno Nettl discusses non-Western notation systems, which 'seem mostly to have or have had an archival or preservative role, perhaps serving as mnemonic devices for performers rather than as aids to composers in controlling and manipulating their structural building blocks' (Nettl 2005: 30). The understanding of notation as rigid prescriptions might be rather recent, as, in the seventeenth and eighteenth centuries, musical scores were often seen as *descriptions* rather than *prescriptions* of music (Haynes 2007); a distinction that can be regarded as the difference between 'composing through performance and composing prior to performance' (Goehr 1992: 188). Furthermore, with developments in the twentieth-century Western musical tradition, notation began to represent the performer's actions rather than sound (Kanno 2007), perhaps strongly influenced by Cage's work with the prepared piano in his *Sonatas and Interludes* (Kotz 2007: 38).

A central concern in musical notation is the idea of repeatability – in other words, the requirement that the composition can be played approximately how the composer imagines it to be played, and that different interpreters perform (or 're-cite') the piece approximately the same way every time they play it. Prior to the advent of phonographic recording this was an important attribute of musical notation, although it should be noted that pre-Romantic composers were

https://doi.org/10.1017/S1355771814000259 Published online by Cambridge University Press

generous in granting interpreters space for improvisation, or extemporisation (Haynes 2007). The scope for creative interpretation by the performer became increasingly narrower up until the mid-twentieth century, when many composers began to create musical work that was more open, often using idiosyncratic syntax, graphic scores or textual instructions. This might result from the advent of phonographic techniques and related electronic music devices, as it could be argued that phonography affected composers in ways comparable with how photography freed painters from the goal of objective representation (Bate 2009: 133). In short, in this new context the objective might not be optimal control over the performer any more, a view exemplified by Roberto Gerhard, who took delight in musical notation's lack of precision:

> Notation's ambiguities are its saving grace. Fundamentally, notation is a serviceable device for coping with imponderables. Precision is never of the essence in creative work. Subliminal man (the real creative boss) gets along famously with material of such low definition that any self-respecting computer would have to reject it as unprogrammable. (Gerhard, in Cage 1969: 240)

The criterion of repeatability as a necessary feature of the score could be contrasted with another principle, namely the score's catalytic features: to what degree does the score encourage creative thinking and performer engagement? This was a concern that became increasingly prominent in the mid-twentieth century, a perspective explored by Umberto Eco in his treatise on the open work (Eco 1989).

With new possibilities offered by digital media of representing sound graphically, descriptive scores have gained a distinct role, where visual artists, animators, 3D graphic artists and VJs have explored different approaches in visualising musical events. This approach could be called representational notation, as it does not necessarily try to describe the music with the intention of archiving or preserving it – modern phonographic technologies are perfectly capable of that – nor is it a form of prescription of human actions, but rather an attempt in representing aspects of the music, interpreting it or simply making laptop-based musical performance more interesting to watch. Rainer Wehinger's listening score (German: *Hörpartitur*) for Ligeti's electronic piece *Artikulation* has become a famous example of such a representational score (Hugill 2012: 62). Other new media ways of writing sound can be seen as being highly representative, or rather exist in a strong audiovisual circularity, where Xenakis's UPIC software serves as a fine example (Xenakis 1992: 329). Live scoring, live notation, live coding, visual notation, dynamic notation and animated notation are just a few names given to the possible cross-modal strands of exploration afforded by the computer.

3. TIMELINES OF CODE

Code is a peculiar form of notation and computers are peculiar interpreters. They are very precise and hard working, and, lacking consciousness (as yet), they never get bored. The coded system can lie dormant for a long time, but is happy to switch to full productivity at the tick of a clock. It can concurrently parse multiple inputs and render parallel outputs. The system can live in its own closed loop of self-reference or socialise with the outer world. In short, when software runs, it is a process that operates through alternate states of waiting and executing orders. These instructions can be scheduled by the program itself as timed events, they can be issued through communication with other software, by way of incoming data from the network of other computers, or through human interaction. An alarm clock, a calendar pushing a notification, an email or social media update, and a multiplayer game are examples of such modes.

For a programmer, scheduling events in time is only one possible form of software activity of many, and some software, for example most word or image editors, relies entirely on user inputs in its workings. The need for representing code itself on a timeline, with an active playhead that moves through time, is a concern particular to a certain type of work. Such work includes generative music and film, interactive games, and other forms that emphasise narrative and carefully designed expression that develops in time. Since creating interactive or generative time-based artistic content involves repeated cycles of coding, testing, evaluating and redesigning, it can be helpful to work with a timeline where a playhead can be dragged back and forth to explore a certain functionality of the code or a particular part of the piece. This is different from the development timelines visualised in clients for various code revision systems, such as Git, where a movement through time is a shift between versions of the software, as opposed to changing positions within a piece, or internal states in the narrative timeline of code scores.

Narrative timelines have typically been created for software development featuring artistic or educational content whose user exploration is an interactive journey. Figure 1 is a screenshot of Macromedia Director, which was an early such audiovisual production environment with an embedded scripting language, focusing on CD-Rom development. Later Macromedia Flash introduced similar functionality for interactive programming of web content. Both were coding environments based on a timeline with frames, where code could be inserted to be evaluated at specific moments in time, much like a musical score. The possibility of adding code into timelines of compositional software is becoming more common, with implementations such as the animation timeline in the

https://doi.org/10.1017/S1355771814000259 Published online by Cambridge University Press

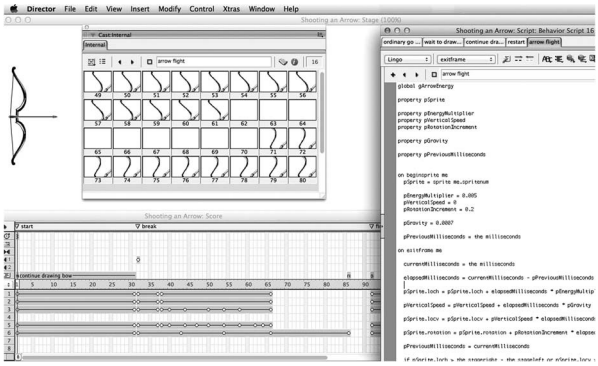

Figure 1. A screenshot of Macromedia Director showing the frame-based timeline, the cast of media elements and a code window. © Gordon Davies

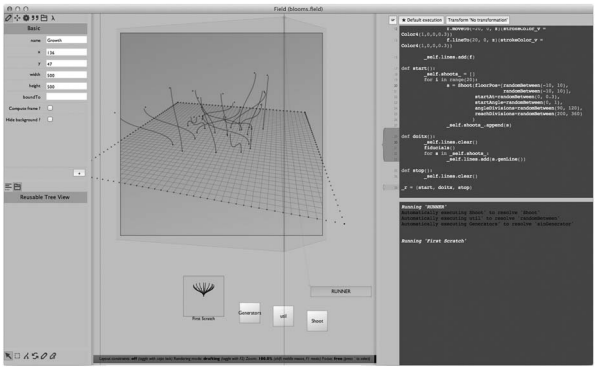

Figure 2. A screenshot of Field: a research animation by Nick Rothwell for the Shobana Jeyasingh Dance Company. The figure shows an interface split in three: on the left a property editor of selected items; in the middle at the top, a rendered graphic box displaying the output of its code, blocks representing other code below, and a vertical timeline that moves from the left to the right triggering events; and finally, on the right, the textual editor for the code with a console output below. © Nick Rothwell.

Unity game engine, the Scripter functionality in Logic Pro X, or Max/MSP within Ableton Live. Furthermore, the idea of scrubbing through the functionality of code using a timeline has been explored as parts of some programming frameworks, for example in the demonstrations of Bret Victor (see http://worrydream.com), and more recently as a *Playground* feature in Apple's Swift programming language. However, these are representations of isolated parts of the code that can be monitored using a *timeline assistant*, and not a general timeline with multichannel tracks where coded events take place in time.

The coding timeline typically affords a strong narrative dimension through an object-focused design, where multiple objects with different parameters can exhibit similar behaviours but have different identities. This can be seen as resulting from the object-oriented programming paradigm, where objects can inherit behaviours and properties from other objects in a hierarchically structured system. In such a system, multimedia elements can be assigned behaviour through code, offering fast prototyping and development in areas of interactive design, storytelling and object-centred content. Indeed, we can detect a change in aesthetics in the digital arts when Processing, Open Frameworks and Cinder became more popular than the named timeline environments. Those coding environments have no timelines by default, typically resulting in work that is loop-based pixel or vector manipulations, often applying impressive artificial intelligence and artificial life techniques, but minimising the multimedia-type narrative content that often characterised work made in the timeline coding environments. Such developments and changes in style are natural and particularly interesting from the perspective of discerning how technology influences artistic expression.

The desire for timelines is strong, and third-party systems have been created for Open Frameworks, Processing, SuperCollider and Max/MSP, and projects like Field (see http://openendedgroup.com/field, and illustrated in Figure 2) operate with coding timelines by default. Timelines are notoriously difficult to maintain, and design questions in such systems include: which parameters should be exposed for control, what level of resolution should be offered in terms of the placement of code (e.g. frame, millisecond or sample), and how is consistency maintained between objects that are created at one point but changed (updating the state or behaviour, for example though live coding) during the execution of the project? Below I will discuss the design of a code score within a compositional system for microtonal performance called *Threnoscope*. However, first I will present the piece from a general perspective, involving a discussion of its representational score.

4. THE THRENOSCOPE

The Threnoscope is a composed system for musical expression within which the performer has a defined scope for improvisation and exploration. The primary mode of performance is by means of live coding, where musical events are created, shaped and terminated through the textual interface of code. A bespoke microlanguage has been created for this purpose, on top of (and extending) the SuperCollider language. The design aims to rethink the emphasis of the linear timeline from musical notation, representation and performance, and rather to present something in the direction of a still image – a stasis on a circular score. Figure 3 is a screenshot of the Threnoscope graphic score. The lines crossing the circles represent an eight-channel surround-sound setup where each line

https://doi.org/10.1017/S1355771814000259 Published online by Cambridge University Press

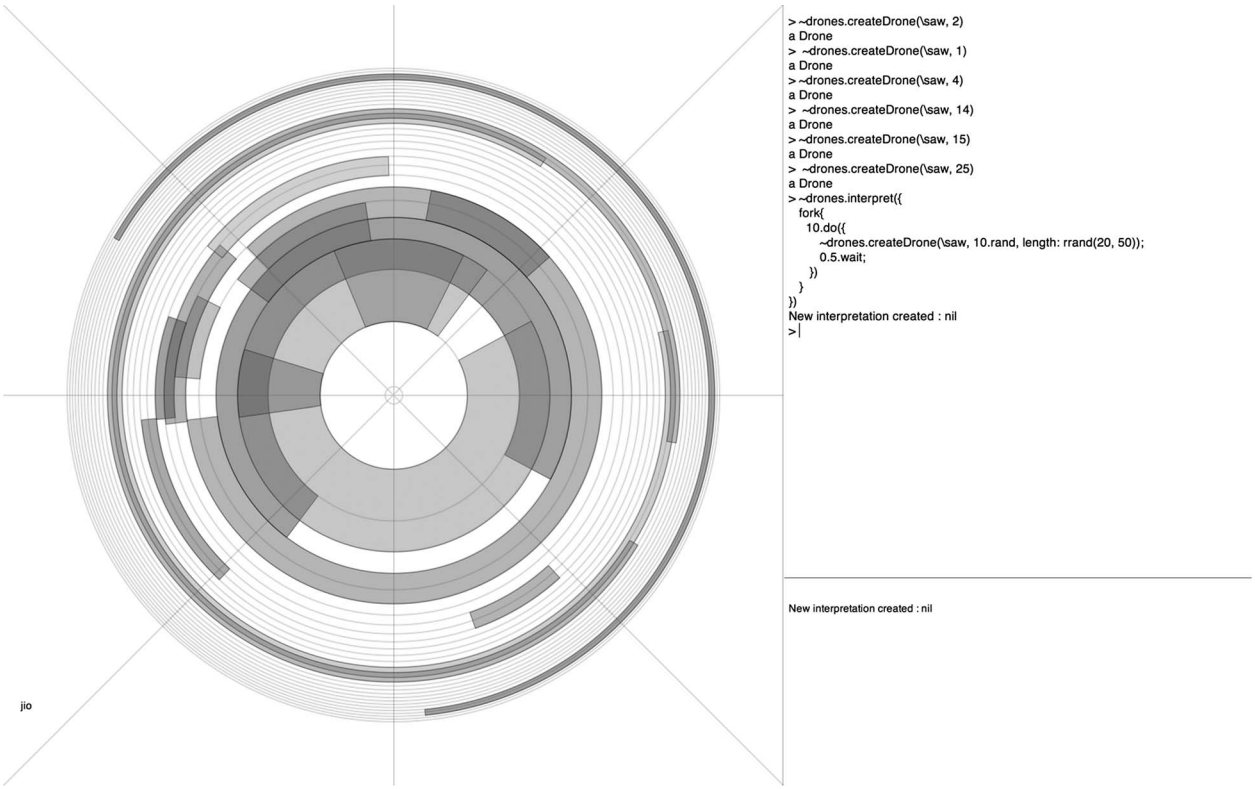

Figure 3. A screenshot of the Threnoscope in its harmonic-view representation. The crossing lines are speakers and the circles are harmonics of the fundamental pitch. On the right there are two text windows, a terminal for code input and a console for system state reports and feedback.

points to the location of the speaker. The half-moons or annular wedges are notes – or rather drones of different waveforms – that can be fully controlled through live coding, graphical user interfaces, hardware, a network or a code score. When a drone enters the speaker line, it begins to sound from that spatial direction, with an attack depending on the drone's envelope. The drones can have perimeter length from 5 to 360 degrees and move around the circular interface at any speed, changing spatial position in the surround sound system.

The circles aligned logarithmically around the innermost circle represent the harmonics of the piece's fundamental pitch. By default this is an A, or 55 Hz, but this can be set to any frequency. The second circle therefore defaults to 110 Hz, the third to 165 Hz, fourth to 220 Hz, and so on. Drones can be created on any of the harmonics, on defined tuning ratios or scale degrees, or by using cents or specific frequencies. The Threnoscope is designed for microtonal exploration of harmonic relationships, including the study of scales, tuning systems and chords. The performer can switch between harmonic-view or scale-view representations of the pitch space, or choose to have both drawn at the same time. Figure 4 shows the Threnoscope interface where a scale-view has been drawn without the harmonics. The system supports the Scala format of scales (www.huygens-fokker.org/scala/scl_format.html), whose library contains thousands of different scales and tunings, including non-octave repeating scales such as the Bohlen–Pierce scale. Scales and tunings can be created in real time and saved in the Scala format. During the creation of scales, hardware can be used for input and playback, for example using a MIDI keyboard, although such hardware might not be the best choice, for example, if working with non-equal tunings or tunings with number of notes in an octave where the arrangement of the piano interface is not suitable.

The drones can be of any waveform, including samples. A typical saw wave at the fundamental frequency (e.g. 55 Hz) would of course contain energy of all the harmonics drawn on the score interface. The saw wave is a useful waveform for exploring tuning systems, as it is often at the higher harmonics where one perceives detuning in harmonic relationships (for example when exploring the difference between a just intonation fifth and an equal tempered fifth). However, distinctive waveforms can also be designed where partials at any amplitude can be inserted above the fundamental frequency, not necessarily at harmonic intervals. This simplifies the design of metallic inharmonic sounds and harmonic sounds with stretched or compressed harmonic intervals, as often found in acoustic instruments. This nuanced design of the drone itself can equally take place

https://doi.org/10.1017/S1355771814000259 Published online by Cambridge University Press

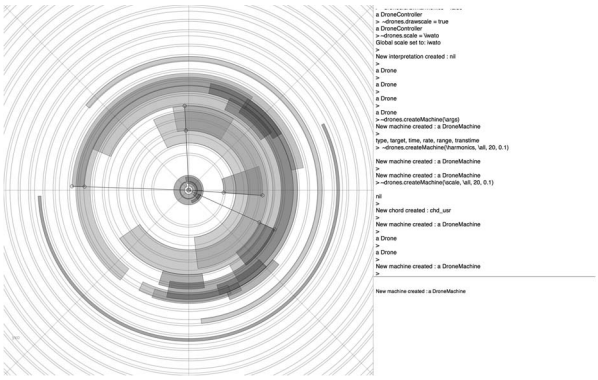

Figure 4. A screenshot of the Threnoscope in its scale-view representation. The figure also shows a drone machine in the middle with arms operating on the drones.

during or prior to performance, as the drone's sound will be compiled and written to disc as a SuperCollider synth definition, and therefore available in the next session (McCartney 2002).

The drones contain a resonant lowpass filter where the cutoff is represented graphically by the width of the drone and the resonance frequency by a line within it. The waveform of the drone is represented by colour, and amplitude by transparency. A drone can be selected for various purposes, and selected drones are indicated by a more prominent and thicker stroke. These can be affected directly by hardware or GUI (graphical user interface) interaction; a simple example is selecting a drone by clicking on it, drag it around with the mouse, or kill it by hitting the delete button on the computer's keyboard.

The Threnoscope contains various intelligent machines that appear in the middle of the circle, as seen in Figure 4. These machines automate processes and support the performer, who can delegate all kinds of designed behaviours to the machines – such as changing harmonics, pitch or amplitude – or create new drones according to relationships detected in existing drones. The machines support a more dispersed and concurrent performance, since they are temporally extended tasks that run parallel to the performer activity. The goal is to create a system where both humans and machines can exert simultaneous control over the system, perhaps reminiscent of the multi-task performance bandwidth of various acoustic instruments as opposed to the single-task focus of a typical live-coding performance. The Threnoscope aims for varied ways of playing, as code, GUI and hardware can be used successfully as control inputs, each mode affording specific features of interaction. As an example, typical GUI elements can be useful for quick tasks, but they suffer from over-simplicity, low resolution and dominating screen presence. Code, however, can be a wonderful interface for musical expression, as complex tasks can be written very quickly, any parameter can be controlled at any resolution, and temporal mechanisms can be set into motion that run on their own, adaptable by the human performer or by the code itself. Defining an action that has an extended duration, with intervallic events, often rendering indeterminate results, is a performer modality that is best achieved by writing code.

The circular score has multiple functions. One of the key objectives of representing drones this way is to enable the performer to keep track of the created sound objects and get visual feedback when they are altered. The textural information present in a microtonal drone piece can be highly complex, and visual representation of the sonic texture – such as overlapping drones, their spatial position, filtering and amplitude – can be very helpful for the performer who wants to add, delete or change parameters in the sonic texture. The representational score also benefits the audience, who can follow events taking place in the piece, for example observing how a drone's range of harmonics is increasing over time, how the drone's fundamental pitch is drifting, or how it moves through space. From audience reports, the visual representation seems to be an important element of the piece, some claiming that the piece should be conceived of as an audiovisual work, where the score cannot be separated from the music. Finally, as a descriptive score, the Threnoscope allows musical collaborators to better understand the ongoing events and improvise with the support of the graphical score.

5. THE CODE SCORE

The Threnoscope was initially created for live coding control, with possible extensions of other control mechanisms. The interface to the complex synth definitions is through method calls, which can be mapped to hardware- or GUI-control in real time. However, it quickly became clear that the system also lent itself to the composition of scores that could be played as parts of a performance or be written as fully composed pieces. A score is here seen as a description of a meta-event that develops over time through a series of sub-events. Such a score could range from a few seconds to hours of activity. The feature that a score can be played at the same level as a note or a chord is uncommon in musical instruments. We do find composed events in many electronic instruments, and some acoustic instruments, such as arpeggiators or chords being triggered by pressing a key or a button, but the current score format allows for a more nuanced composition of such events, including generativity.

The Threnoscope contains a score-playing mechanism that executes code at specific moments in time. The score is in a textual format and can be written manually using a text editor or by the system itself, as the code history of every session is recorded by default. The score format is a two-dimensional array, containing information about points in time and designated code

https://doi.org/10.1017/S1355771814000259 Published online by Cambridge University Press

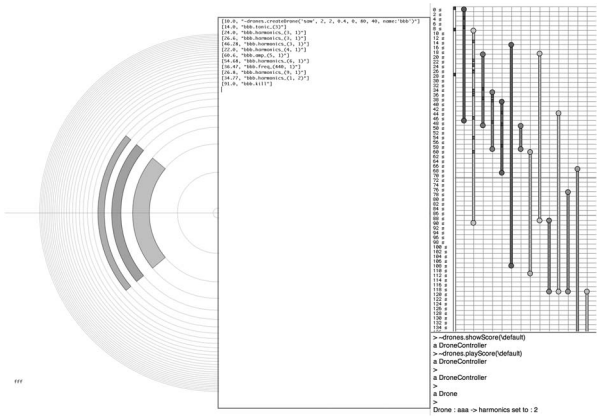

Figure 5. A screenshot of the Threnoscope code score. Above the representational score is a text field with the code for a selected drone. At the right there is a visual representation of the whole score.

to be executed. Multiple scores can be played in parallel, as each contains its own timing mechanism and local variables.

Whilst the textual format has its positive qualities, it does not contain a visual representation of the timeline. Such visual rendering can be helpful for comprehending the structure of musical pieces. A graphical timeline system for the code was therefore implemented. Initial experiments included representations of drone parameters such as amplitude, harmonics or speed, but since code can theoretically generate anything, this was seen as limiting. A more fruitful solution was to create a 'code score' where code itself is visualised (rather than its output) on tracks for each drone, accompanied by one global track for system-wide code instructions. Figure 5 shows the code score on the right. The global code track is on the left, where any SuperCollider code or system-wide commands can be placed. System commands include the type of tuning or scale to be used, the type of visual representation, and so forth. The other tracks visible are the drones themselves, with start and end points, and drone-specific code is indicated with a small rectangle in the drone timeline itself. The score elements can be moved back and forth during playback, enabling the performer to operate on the score in real time. Events can be added to the score during playback by both human and code intervention.

The code score is an attempt to create a dynamic and adaptable timeline in a musical system. Each drone-specific instruction is represented by a small square on the timeline. The composer thus gets a clear visualisation of the piece's form, but also gains a fast method of arranging events, familiar to those accustomed typical MIDI editors or digital audio workstations. In this manner, drones can be dragged around, extended, and events within them moved. As a narrative timeline (formal-timeline), the code score exists as a part of a short but interesting tradition that has explored the possibility of representing code and objects affected by code on a timeline with a moving playhead. Often such timelines are frame-based, running through a certain number of frames per second, but the Threnoscope score is not based on frames, but temporal events, supporting a time resolution down to the sample level.

From this perspective, it is clear that live coding with the Threnoscope is adopting a pre-Romantic method of 'composing through performance' (Goehr 1992: 188) where important aspects of the piece are composed through live performance yet others reside in the system's design. Live coding extends this practice by also enabling the instrument and the score to be designed and altered in real time. The relationship between improvisation and execution of a score in the Threnoscope can be interesting, as the score can range from being very simple and short to a highly nuanced piece of longer duration, but one that can be manipulated at all levels in real time by the live coder. In this case the performer improvises or plays on top of a prior design, not unlike certain practices in instrumental music – for example Steve Reich's *Electric Counterpoint,* where a performer plays on top of a prerecorded performance.

6. CONCLUSION

This article has developed the idea of computer code as a form of musical notation that might benefit from a linear representation of time. It has argued that the timeline representation can be an important element in musical composition and subsequently explored graphical code scores in that context. Since live coding is essentially an improvisational practice, most live coding systems do not implement scoring mechanisms besides code itself. However, through the possibility of arranging code events on a timeline using a bespoke format, the live coder can simplify the compositional thought process, as the piece (or the relevant part of it) can be externalised and represented both textually and graphically on a timeline. Such external representations allow for more abstract compositional arrangements, higher levels of dynamic scoring, and better support for real-time composition.

The article described the current state of the Threnoscope as a performance system implementing the textual interface as a score mechanism, in addition to two types of graphic scores: the representational score of the system's internal state, useful to both the performer and the audience, and the prescriptive code score that enables the integration of designed event patterns with live performance. The code score is a type of musical event integral to the instrument, equal to the drones or machines that can be played at any time. It is not an external system for denoting actions, but rather a feature of the instrument, like a note on a keyboard. Since the Threnoscope was originally conceived of as a musical

https://doi.org/10.1017/S1355771814000259 Published online by Cambridge University Press

piece with an embedded micro-language for live coding performance of microtonal drone music, its scope was more limited and narrow than most live coding systems, thus granting the direct explorations of descriptive scores and code scores. The code score in the Threnoscope is proposed as a solution to the problem of how code can be presented on a time line axis, given that code is itself its best representation.

The author has performed with the Threnoscope at festivals and conferences and has received numerous requests by people who want to use the system in their own musical practice. This raises interesting questions as it is still unclear whether the system should be seen more as an instrument for musical expression or as the musical piece it was originally intended to be. Questions of authorship and collaboration are relevant, but set aside for further study, together with exploration of acoustic instrumentalists performing against the representational score and systematic studies of audience feedback.

Acknowledgements

I would like to thank Dr Stephen Boyd Davis for the fruitful discussions over the years about timelines and representations of events in time. I am also greatly indebted by the profound comments and suggestions provided by Dr Ed Hughes, Gordon Davies, Enrike Hurtado Mendieta and Birta Thrastardottir.

REFERENCES

Bate, D. 2009. *Photography: The Key Concepts*. Oxford: Berg.

Bowker, G. C. and Star, S. L. 2000. *Sorting Things Out: Classification and its Consequences*. Cambridge, MA: MIT Press.

Boyd Davis, S. 2012. History on the Line: Time as Dimension. *Design Issues* **28**(4): 4–17.

Clark, A. and Chalmers, D. J. 1998. The Extended Mind. *ANALYSIS* **58**(1): 7–19.

Collins, N., McLean, A., Rohrhuber, J. and Ward, A. 2003. Live Coding in Laptop Performance. *Organised Sound* **8**(3): 321–30.

Eco, U. 1989. *The Poetics of the Open Work. The Open Work*. Cambridge, MA: Harvard University Press.

Goehr, L. 1992. *The Imaginary Museum of Musical Works: An Essay in the Philosophy of Music*. Oxford: Oxford University Press.

Good, M. 2001. MusicXML for Notation and Analysis. In W. B. Hewlett and E. Selfridge-Field (eds), *The Virtual Score: Representation, Retrieval, Restoration*. Cambridge, MA: MIT Press.

Green, T. R. G. and Petre, M. 1996. Usability Analysis of Visual Programming Environments: A 'Cognitive Dimensions' Framework. *Journal of Visual Languages and Computing* **10**(2): 131–74.

Haynes, B. 2007. *The End of Early Music: A Period Performer's History of Music for the Twenty-First Century*. Oxford: Oxford University Press.

Hugill, A. 2012. *The Digital Musician*. 2nd edn. Abingdon: Routledge.

Kanno, M. 2007. Prescriptive Notation: Limits and Challenges. *Contemporary Music Review* **26**(2): 231–54.

Kotz, L. 2007. *Words to Be Looked At: Language in 1960s Art*. Cambridge, MA: MIT Press.

McCartney, J. 2002. Rethinking the Computer Music Language: SuperCollider. *Computer Music Journal* **26**(4): 61–68.

McLean, A. 2008. Live Coding for Free. In A. Mansoux and M. de Valk (eds), *Floss + Art*. London: Mute Publishing.

Nettl, B. 2005. *The Study of Ethnomusicology: Thirty-One Issues and Concepts*. Champaign, IL: The University of Illinois Press.

Seeger, C. 1958. Prescriptive and Descriptive Music-Writing. *The Musical Quarterly* **44**(2): 184–95.

Sorensen, A. and Brown, A. 2007. aa-cell in Practice: An Approach to Musical Live Coding. *Proceedings of the International Computer Music Conference*, Copenhagen/ San Francisco: ICMA. 292–9.

Wright, M. 2005. Open Sound Control: An Enabling Technology for Musical Networking. *Organised Sound* **10**(3): 193–200.

Xenakis, I. 1992. *Formalized Music: Thought and Mathematics in Composition*. Hillsdale, NY: Pendragon Press.