Live Coding

A User's Manual

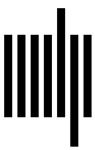
By: Alan F. Blackwell, Emma Cocker, Geoff Cox, Alex McLean, Thor Magnusson

Citation:

Live Coding: A User's Manual
By: Alan F. Blackwell, Emma Cocker, Geoff Cox, Alex McLean, Thor Magnusson
DOI: 10.7551/mitpress/13770.001.0001
ISBN (electronic): 9780262372633
Publisher: The MIT Press
Published: 2022

OA Funding Provided By:

OA Funding from Author



The MIT Press

4 Notation

Having mapped out some of the histories and contemporary examples of live coding's evolution as a performance practice, the emphasis in the following chapters shifts to address how live coding opens up critical issues relating to liveness, temporality, and knowledge production, as well as the notion of notation. This chapter focuses not only on notation but on what is notated and the activity of notation—the nature of the algorithms that live coders work with and the dynamic ways in which they are crafted. By examining how live coding challenges tensions between liveness and determinism, between musical improvisation and composition, and between oral and written culture, we find new approaches to notation as a dynamic, live medium.

As the field of live coding expands, embraced and developed by other disciplines beyond the original traditions of computing and music, the understanding and application of notation practices and principles have also changed and transformed, bringing both the potential for the hybridization of the concept of notation as well as a risk of confusion arising from the lack of shared conventions or vocabularies. However, rather than argue for an agreed definition and application of notation within the field of live coding, we explore if live coding can itself operate as an exploratory site of interdisciplinary exchange wherein the concept of notation is roughened and problematized. The chapter begins by highlighting two research projects that have explored notation within live coding in relation to other disciplines before looking more closely at live coding notations and how they complicate the notion of notation itself.

Live Notation

Initiated in 2012, the research project Live Notation: Transforming Matters of Performance (2012) was established in order to examine the shared vocabularies that may unite two performance practices[1]—namely, live coding (performing with programming languages) and live art (performing with actions). From the perspective of the

live coders involved, this allowed them to examine their practice with fresh eyes, not in terms of what was "new" in a technological sense but what was commonly shared with another, well-developed performance art practice. Certainly, for many live coders, liveness and risk are at the core of the practice. The challenge is not about performing prewritten code; rather, the performance emerges in and through the liveness of the event itself: through a relationship with the audience, other performers, the room acoustics, the previous and following acts, and the adrenaline of the live, which all shape the performance and the experience of performing. Likewise, for many live art practitioners, *liveness* refers to the durational, embodied, nonrepeatable moment of performance. Organized as an "experimental laboratory," the Live Notation project attempted to "approach programming as performance art, performance art notation as code, code as speech, bodies as interpreters," involving "improvisational sound works (where computer code and the artists' bodies become instruments), site-specific time-based art works (where notation becomes the 'piece' as opposed to its recording device)" alongside a series of position papers.[2] An attempt was made to challenge or disrupt the function of notation as that which either precedes performance (as a score or script written in advance and executed live) or that follows in the form of a recording or document of a performance (supporting its future reactivation or replaying) by testing forms of practice where the notation is produced synchronously to performance itself. Reflecting on the specific practices encountered within this project, the term *kairotic notation* (drawing on the Greek term *kairos*, "opportune timing," to be discussed in chapter 6) was proposed by Emma Cocker as a way of articulating the distinctiveness of live notation from simply the performance of notation live.[3]

Live notation, or rather kairotic notation, refers explicitly to practices (including live coding) in which a form of notation is produced as a live event simultaneously (and in fidelity) with the experience it attempts to articulate. Here, live notation is composed in front of the audience through its performance, unlike conventional forms of scripting for performance that are *decomposed* or that disappear as they are performed, as poet and essayist John Hall asserts.[4] Live notation is the kairotic or kairic event of creating an adequate form of articulation simultaneous to the experience or ontology that it attempts to describe. The performance produces its own score, during itself. Live coding is performed as a *recursive loop*, where "notation and execution are collapsed into one thing," breaking down the "false distinction between the writing and the tool within which the writing is produced."[5]

The Live Notation project also drew attention to many aspects of performance that are often ignored or remain invisible within conventional notational languages: those embodied, experiential, intersubjective vitality forces and affects operating before,

between, and beneath the more readable (therefore arguably more writable, inscribable) gestures of a practice. Likewise, through the research project Weaving Codes, Coding Weaves (2014–2016),[6] live coders Alex McLean and Dave Griffiths joined weaver and mathematician Ellen Harlizius-Klück to address the challenges and deficiencies of conventional notational systems for describing complex embodied procedures through exploring the relationship between ancient weaving and computer programming. This project asked, among other questions: How might the complex interwoven procedures of ancient weaving be addressed through coded algorithms, when the tendency in digital rendering of weave is often one of attending to and defining a discrete (isolated) operation or function? How can computational notation accommodate the possibility of two or more weaving techniques within the same fabric? Can computational notation capture and communicate the sense of the tacit knowledge necessary for weaving, the critical deliberation, and the tactile and embodied processes of trial and error in weaver's work with the resistances, tensions, and even unexpected surprises of both the loom and thread? Moreover, how can notation articulate the sense (and value) of the decisions made "on the loom" so central to ancient weaving?

Central to the Weaving Codes, Coding Weaves project was an attempt to dislodge the privileged model of "working out" when an idea is applied to material (having been conceived in advance), in favor of a model wherein various levels of operation and cognition are activated live within the process itself. What emerged through this research was a sense of the complex, combinatorial properties of ancient weaving, which renders any single system of notation or simulation inadequate. The weaver works with multiple notational languages at the same time, live weaving them together as a singular experience or even gestalt. Additionally, different systems of notation can illuminate or privilege different facets of the weave process, in which the tendency is often one of attending to the operational settings of the loom (the heddles, the lift plan) alongside the notation of the product—the resulting weave structure—itself, rather than the temporal, tactile, and even sensuous movements of either the weaver or the thread.

In one sense, the Weaving Codes, Coding Weaves project made tangible that which conventional notational and simulation languages fail to account for but the weaver knows all too well: the importance of timing, timeliness, tension, rhythm; the negotiation of different and even competing forces within the process of weaving; the tactility of a weave's three-dimensionality; and the textural properties of thickness, roughness, density, and stretch. Both ancient weaving and live coding involve a live and embodied process of decision-making that operates in excess of, or perhaps even between, the lines of conventional notational systems. Within each practice, there is a sense of oscillation or even "shuttling" between discontinuous systems of abstract notation and the

continuous experience of a lived process and between the importing of source codes and preexisting patterns and a mode of invention that actively modifies the process as it unfolds. The tensions between the abstract and discontinuous logic of notational systems and the ways in which they become reembodied through practice gets picked up in chapter 5 with explicit reference to the notion of liveness—when the experiential liveness of the performer meets with the technological liveness of the machine.

The Notion of Notation

These two research projects identify commonalities and resonances between live coding and other practices (specifically, live art and weaving). They consider what happens when musical and computational conventions and understandings of notation meet with notational practices developed within other disciplines, including dance, performance and the visual arts, and textile arts, such as weaving and braiding. In so doing they draw attention to how the term *notation* resonates with different meanings and values within different disciplinary traditions, inviting reflection on what the implication of this is for the future of live coding practices. Within the frame of interdisciplinary research and collaboration, is there any real consensus on what is referred to by the term *notation*? Accordingly, this chapter sets out to move from a general notion of notation to address the specific questions that live coding raises as a notational practice.

Notation can operate in different ways within various disciplines, ranging from colloquial use for various note-making practices, to other forms of score, script, recipe, or diagrammatic map, to the development of a formalized notation system with its own clearly defined inner logic. Notation involves the production of marks or symbols, the generation of signs relating to a signless experience. It operates within a semiotic field: What or how is the relation between sign and signification? In one sense, notation is activated whenever a sign or mark is used to stand for—represent. Addressing the problem of notation for interactive media—or the "dangerous quest for a media art notation system"—researchers Simone Boria et al. put notation forward as "an abstraction, a simplification, and intuitive or studied way of writing something down that succinctly summarises the important points of a given situation, process, object or system."[7] They further argue that "a system becomes a notation system when it has a working inner logic using a set of abstract presentations (vocabulary) of aspects of potentially universal experience deemed relevant to be differentiated between, preserved and communicated about."[8]

Boria et al. elaborate the criteria for "notation-system-ability" thus: "Is there an inner logic? . . . Is there a vocabulary? . . . Are the notations potentially accessible to at least one

entity/person? . . . Are other aspects intentionally left out."[9] Additionally, they conceive a general catalog of notation systems that incorporates the following categories: gestural notation—cheironomy or the use of hand signals; scientific notation—the abstractions of mathematicians, physicists, and so on; musical notation—with its associated ideas of score, composition, and interpretation; dance notation—with its genealogy from the pictograph methodology to the real-time one-to-one "notation" of video recording; painting notation—or perhaps, more broadly, a conceptual art tradition involving the principles of instruction and execution while willfully minimizing expressivity; spatial (as in nontextual) notation—maps, data visualizations; computer notation—involving writing code and its execution.[10] Boria et al. identify a spectrum of purposes for notation rather than any singular function—namely, to understand, to navigate, to share and archive, to engineer, to analyze, to interpret, or to disguise. Within the expanded frame of live coding, how can these different systems, traditions, and purposes of notation be investigated in and through practice?

Notation History and Change

To address the specific questions that live coding raises for the practice of notation, we begin with a brief account of the evolution of music notation, which has its origins in early writing systems. Some of the oldest historical articles of musical notation are Hittite tablets, from Ugarit in today's Syria, written in cuneiform some thirty-four hundred years ago. The ancient Greeks had notational systems, too, but most ancient musical writings were written in the form of theory. This makes sense in the age of scarce media, as a theory of music is generative and can produce practically infinite versions of music. Western classical music developed very different ideas of authorship and performance, largely deriving from Romantic ideas about the roles of composers, the musical work, and its interpreters. Medieval monks developed systems for describing intervals, called *neumes*, written above the sung text, but this was more of a memory aid than prescriptive notation. With Guido d'Arezzo, an Italian monk of the eleventh century, we begin to see revolutionary ideas of musical notation. Guido invented the staff notation still in widespread use today, claiming that finally music could be performed by people who had never heard it before.[11] The development of notation in the following centuries represents a rich and exciting history, but from a media theoretical perspective, a drastic change appeared with printing.

Following the Gutenberg press in the fifteenth century and stimulated with the musical culture of the baroque, the primary purpose of staff notation in the West was the documentation, composition, and distribution of music. Early printed works had

plenty of scope for interpretation, improvisation, and extemporization, but written notation became increasingly complex in the twentieth century, partly following developments in print technology. Many of the things twentieth-century composers wrote would not be possible to convey in earlier movable type musical notation and engraving technologies. With new forms of print and graphic reproduction technologies, new modes of notation, such as graphic notation, became easier to work with.

While staff notation has had international influence on the world of music, it is rarely seen in live coding music communities. One reason for this is that a primary motivation for live coding is to engage with music through computational processes, and while staff notation is representable as a symbolic, discrete data structure, it is hardly a computational one. With some exceptions, such as ossia, there is no scope for logic or branching in staff notation, meaning that it is not possible to express any possible algorithm (i.e., not *Turing-equivalent*, in mathematical terms). It is certainly possible to "live code" an instrumentalist, as we find in Magnusson's *Code Music Notation*, created for a collaboration with marimbist Greta Eacott, who took the role of the interpreter in a performance at the International Conference on Live Coding in 2015.[12] In this performance Magnusson wrote notation for the performer in a human-readable algorithmic language based on machine language. Musical scores can, of course, be changed in the middle of a performance,[13] but any computation in the process would happen elsewhere, perhaps to generate that score.[14] Many projects are algorithmic and live in nature, but the typical approach of live coding notation is to adopt the language of the computer itself: the programming language.

Another reason for the lack of staff notation in live coding is cultural. In the West, live coders most commonly draw from urban club and dance music usually originating from the African diaspora, such as New Orleans jazz, Detroit techno, Chicago house, hip-hop from the Bronx, reggae from Jamaica, and jungle from England, as well as electroacoustic, "experimental," and noise musics from alternative and academic computer music practices. All these influences could be considered to be oral cultures, where musical techniques are mainly shared not as formal notations but through demonstrations and word-of-mouth. Indeed, while live coding practice is heavily centered on writing code, it can nonetheless be argued that live coded notation, with all of its ephemerality and impermanence, has features that are closer to speech than to writing. Live coding practice therefore finds itself caught between two worlds: it is too ephemeral to be score-based culture and yet too centered on text to be oral culture.

We return to the question of speech versus notation later in this chapter. In any case it is undeniable that live coding is a supremely notational practice, in which (with some exceptions, such as in live coded choreography) every event arises from notation

with explicit, formal meaning. Indeed, live coding notation is even more explicit than staff notation, whereby some interpretation is generally left to the instrumental performer; by contrast, code is by nature deterministic. What breaks this determinism in live coding is that the code itself is open to change at any point, bringing the computational processes of the machine and the thinking processes of the coder together in a single cognitive loop.

Returning to our historical timeline, the nineteenth century brought the invention and development of the pianola or player piano, for which music could be bought on paper rolls, with notes played via holes punched in the paper, driven by pneumatics. From today's perspective, it is interesting to note that player pianos (as opposed to *reproducing* pianos) were not fully automatic but were commonly designed with expressive controls for tempo and dynamics for a human operator to "play."[15] Indeed, there were once virtuoso concert pianola players, most famously Rex Lawson, who played music composed for the pianola, as well as Igor Stravinsky and Conlon Nancarrow.[16] Some rolls were not only punched with holes controlling the notes but were also printed with lines directing the human player pianist how to work the hand controls in order to fully reproduce the performance of a particular human pianist or even the original composer. Although the naming of a performer may have been more of a marketing device than an accurate recording, the printed lines are a clear acknowledgment that as a notation, the punched holes as notes within a grid of such metronomic rolls are an incomplete representation of music.[17]

Where it is deterministically interpreted, live coding notation, on the other hand, is a complete representation of the media currently being produced by it and therefore sits in its own category: too transient and in the moment to be considered a recording and too complete to be considered a mnemonic notation. But perhaps rather than standing for a brand-new way of thinking about music, live coding instead exposes flaws in how we have, in recent history, come to think about music as an end product, rather than as a live activity.

Live coders perform with code, but it does not follow that the performance is itself coded. That is, live coders embody their code and think through it but are not controlled by it. From the outside, live coding culture could be mistaken for being dehumanizing and lacking expression, but this is a misunderstanding. Live coding is about *disrupting* the deterministic logic between notation and process, bringing it into a creative feedback loop where that logic evokes an unconstrained experience, feeding back into edits that are not predetermined.

This is analogous to the Machinery, a traditional clog dance originally from the working-class cotton mills of Lancashire, UK, which mimicked both the sounds and

movements of machines.[18] The Machinery takes the repetitive processes of industrial technology and acts them out in repetitive, jerky movements, creating noisy clattering and scrapes with the clogs, reproducing the sounds and movements of power looms and other machinery. By embodying the machine in this dance, the dancer does not become a machine; rather, machinic movements become human, and the millworkers regain human agency from mechanization. Looking at live coding in this light, we can see that live coders similarly do not become coded but rather *embody* code.

The Machinery began with women mimicking the movements of machines with their clogged feet while they worked. Eventually, it was brought out of the mill and into the world of mainstream culture and performed without the real machine present. Could the same thing happen with live coding? In the algorithmic dance world, it already has, with choreographers such as Kate Sicchio notating instructions not for computers but for human dancers (see her exposition in chapter 3). Without the computer, the difference between performing logical operations and "playing" logic then becomes much clearer. Dancers interpret instructions on their own terms, exercising agency in ways that are not predetermined.

Experimental Art Traditions of Notation

John Cage's book *Notations* (1968) is a well-known reference for the art of notation in the postwar experimental tradition. Indeed, there is a rich history across the twentieth century to draw upon, echoing experimental practices in language such as OuLiPo, mentioned in chapter 1, the instruction pieces of Fluxus, and avant-garde performance practices more broadly. The connections between notations, programs, and scripts underpin our concerns with how the structures that generate movement are made visible—both readable and writable—and how aesthetic and functional perspectives conjoin in graphic scores and executable forms. Often foregrounded in this history is the way in which scores can be used to generate indeterminacy, as a way to navigate a space of possibilities—for instance, the commonly cited *chance operations* of Cage and the dice games popular among Western eighteenth-century composers.[19]

Another common reference is Luigi Russolo's manifesto *The Art of Noise* (1913),[20] in which all sounds can be considered musical and therefore demand new forms of notation. The Fascist politics to which the Italian Futurist movement related, and of which Russolo was part, should, however, give us pause for thought in their call to discard history. The influence of the visual arts is also felt in the example of Wassily Kandinsky and Paul Klee, in particular, in relating structures and colors in their paintings with rhythm and timbre. Again, Cage—not least in his approach to noise—makes a good example in

his cross-media collaborations with dancer Merce Cunningham in chance operations. To Cage, music comes together with other phenomena in expressing the absence of logic, coherence, and predictability in everyday life. An example from the live coding field that explicitly draws upon this tradition is Nick Collins's *Avscore 37*.[21] It is a graphic score with two channels: player A interprets a succession of framed abstract scenes, and player V follows a continuous staved timeline. Together the score acts as a *suggestion* for how the audio and visual elements might correlate or not. The score itself is computer generated and so folds back onto itself in expressing the creative possibilities of its live performance.

The foregrounding of notation in the form of a code or rules extends the legacy of Fluxus scores and conceptual art instructions and the prevalence of algorithmic procedures within computer art, where code or rules become generative strategies for producing outcomes potentially autonomous of artistic control or agency. Recall the Fluxus performance score of La Monte Young's *Composition 1961 No. 1, January 1* as operating in analogous terms: "Draw a straight line and follow it." Here, as Sol LeWitt remarks, "To work with a plan that is pre-set is one way of avoiding subjectivity," where "all the planning and decisions are made beforehand, and the execution is a perfunctory affair. The idea becomes a machine that makes art."[22] Live coders bring this subjectivity back into view, although given that La Monte Young's composition is so open to interpretation, perhaps it was there all along.[23]

Representation and Style

Antony Braxton broadly divides the world of music practices into three categories:[24] *stylism*, which no longer changes; *traditionalism*, which continues a long history of change; and *restructuralism*, which signals a new kind of music as a break from tradition. It seems that notation has a role in deciding whether a restructuring of music develops into either a tradition or a style. The power of written and printed music (and to an even greater extent, recorded music) is in its capacity for mass production and dissemination, but in supporting the notion of authenticity, or *Werktreue*, individual pieces are less open to structural change. Oral tradition, on the other hand, particularly in folk music, necessarily undergoes change through the act of transmission. But from the perspective of music theory, we could also say that notated music emphasizes change because every piece has its own identity, at times even its own music theory.

When human expression is represented on a computer, whether music, video animation, or choreography, it is reduced to numerical data that are being executed in time. This is most easily seen in grid-based music,[25] in which pitches and durations are given discrete numerical values, typically using the MIDI standard whereby integers on

a linear numerical scale are mapped to the exponential frequencies of musical pitches. Notes can be stored as numbers in lists and chords as sublists. In computer music, timbre is often more important than pitch and can also be represented as numerical values—for example, as synthesis parameters. Relatedly, we can find code libraries for spectral manipulation and machine listening that can organize and manipulate sounds along perceptually salient dimensions, of use for live coding systems. Choreographic representations are perhaps less able to be reduced to grids of numbers, but here computer vision and machine learning may also be applied, such as Sicchio's work in organizing visual material into quality dimensions, creating a space of possibilities that can then be navigated with live coded instructions.[26]

Through notation, then, the live coding performer can engage with any parameter that they judge to be of interest, whether controlling movement, pitch, timbre, or a higher-order rhythmic manipulation. A specific live coding system will afford the control of some or all of these elements, but it is clear from the plethora of available systems that the authors of the systems are not necessarily interested in all. Algorithms are used to describe patterns and shape events in time. The interesting question for artists is *how* this is done because this will inevitably color the output. Live coding systems therefore incorporate methods for their users to generate events over time while continually shaping the result through live engagement with the running program. In the early days of live coding, practitioners would often design their own systems, and the character of the inventor would shine through in the way the system worked. Live coding environments typically supported what the author of the environment wanted to do in a performance. Even today, systems such as Scheme Bricks, Extempore, TidalCycles, Hydra, ChucK, Threnoscope, Foxdot, Sonic Pi, Improviz, and Gibber all exemplify certain views on what is important in live performance, whether that is expressive range, speed of writing, compositional potential, understandability, surprise, timbre, spatiality, rhythmic patterns, visual or melodic progression, and so on.[27]

A live coding language is an environment in which live coders express themselves, and it is never neutral. People who speak more than one natural language are familiar with how language shapes thoughts and personality. A switch to another language might even affect how we move our bodies when we speak. Such effects are especially pronounced in live coding, as the languages are typically high level, potentially designed with particular visual or musical styles in mind and offer particular creative constraints. We can therefore argue that live coding languages inevitably shape the thoughts and actions of the user, but here the user is also a coder. Live coding systems range from being more akin to individual pieces, such as Sicchio's Terpsicode or Magnusson's Threnoscope, to more general-purpose, systems-level programming languages. Where live

coders choose a more general-purpose language to work in, such as Extempore, SuperCollider, Python, Lua, or even the venerable C language,[28] they may choose to simplify the expressive range of the language in order to create their own constraints, providing a more manageable space of possibilities to creatively work within and against.[29] Alternatively, they may extend the environment with additional vocabulary or techniques as a way to forge their own style. This has been called *pre-gramming*,[30] as it involves a preliminary preparation for the live coding practice, a process where language design inevitably merges with musical composition.

Defining Live Coding through Notation

The ideas represented by live coding, live programming, interactive programming, or conversational programming are not new. Indeed, the design of common live coding systems, such as SuperCollider, refers to the cybernetic ideas of Gordon Pask; the live electronics of David Tudor; the programming language design of Alan Kay (Smalltalk), David Ungar, and Randall Smith (Self); and Steve Tanimoto's ideas about levels of liveness.[31] The idea of creating an object or function that can be named and altered while running is appealing for all time-based art forms, whatever the domain (choreography, virtual worlds, robotics, animated graphics, music). What is abstracted and represented is often the work of the language designer, which highlights the compositional decisions involved. For this reason, we have not seen a coordinated effort to build a general live coding language, but many individuals are making their own systems, albeit often released to the public and used by others in studio work and performance.

The notational considerations in live coding systems serve multiple purposes. Designers of the systems have in mind things such as ease of learning, ease of understanding for lay spectators, expressivity, error tolerance, speed of writing/tersity, tracing, manipulation of code history, and many other language-design features that have hitherto not always been considered relevant in traditional programming-language design. This is changing, however, with the (re)emergence of the communities around programming-language experience design and the "future of coding." This resurgence of interest in liveness among software engineers may have been partly inspired by the live coding movement, although interaction designer Bret Victor is much more widely cited as an influence through his well-distributed videos including "Stop Drawing Dead Fish," and the "Future of Programming."[32] Following work with pioneering computer scientist Alan Kay,[33] Victor has since established the independent DynamicLand laboratory, modeled on Douglas Engelbart's earlier Augmented Intelligence Lab, to take ideas around live, tangible, computational environments further.

Algorithmic Pattern

Live coding is an algorithmic art form in that code is written to represent algorithms while those algorithms are being enacted by a computer to generate musical, visual, kinetic, or other live results over time. However, we too often use the word *algorithm* without fully articulating what algorithms are in terms of how they are structured and how they operate. In computer science, an algorithm is often defined as a finite sequence of unambiguous instructions that can be followed in order to solve a problem. This is clear enough but leaves much to say about how those rules are structured and followed.

Magnusson and McLean have elsewhere asked, "How can we directly express musical patterns with computer code?,"[34] examining which strategies the performer can apply in live coding for the patterning of music and which strategies there are at hand for transforming the musical materials. We can algorithmically generate musical data or write them by hand, but live coding does more than that: we apply algorithms to alter these data. The aforementioned text drew upon a 1981 article by composer Laurie Spiegel to frame examples of the pattern methods that can be applied onto musical data or directly used to represent music.[35] Spiegel gives an explicitly nonexhaustive list of twelve categories of pattern manipulation from her perspective as a composer with a foundational role in the development of computer music: 1) transposition, 2) reversal, 3) rotation, 4) phase offset, 5) rescaling, 6) interpolation, 7) extrapolation, 8) fragmentation, 9) substitution, 10) combination, 11) sequencing, and 12) repetition.[36]

From this approach, the basis for pattern in computation (and vice versa) becomes clear, particularly when we consider the lower-level operations of computer machinery. For example, the logical operators "AND," "OR," and "XOR" are instances of combination (combining two values into one), "NOT" is a form of substitution (zero for one and one for zero), and "<<" (left shift) and ">>" (right shift) are forms of rotation. As such, live coder and weaver Dave Griffiths has visualized the state of registers over time while simple calculations are performed by a CPU (figure 4.1) in order to demonstrate the provenance of contemporary computation in ancient textile techniques, particularly weaving and braiding.[37]

From this perspective, computational algorithms and patterns culturally situated in textiles, music, and dance seem closely related. However, this comparison does not sit well in parts of the music field where composers in general use the word *pattern* to describe any fixed sequence, sometimes even in a pejorative sense. Although the centuries-old fugue, as a contrapuntal compositional technique, is based on pattern thinking, some composers would be deeply offended by the accusation that they are

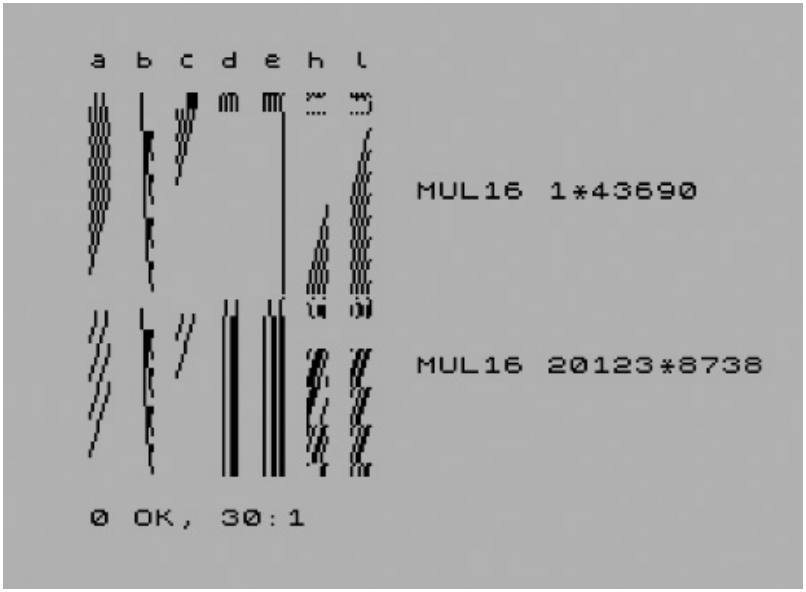


Figure 4.1
Visualization of the eight-bit registers of a Z80 microchip as it performs simple calculations, demonstrating the relationship between computation and weaving.
Source: Computer artwork by David Griffiths.

making mere patterns.[38] Minimalism—notably, the work of Steve Reich in applying a phase operation to a bell pattern in his 1972 piece *Clapping Music*—demonstrates the generative nature of pattern in Western concert halls as additional context for Spiegel's 1981 paper.[39] But the connection between algorithmic patterns and the far longer history of handcraft, particularly textiles, is clear, and we return to the topic of weaving and coding at the end of this chapter.

Since Spiegel's paper, many computer music systems have explicitly included pattern transformation features, from the early Hierarchical Music Specification Language (HMSL), to the Common Music and Bol Processor systems in the 1980s, to SuperCollider in the 1990s and many of the systems developed and used by live coders. The TidalCycles system has perhaps gone furthest in this direction, being designed exclusively for live coding algorithmic patterns. It consists of a *mininotation* for rhythms, heavily inspired by the Bol Processor's polymetric expressions,[40] and an extensive library of combinators offering a wide range of possibilities for manipulating pattern. The live coder is free to combine any number of these functions together, providing a very rich range of possibilities for patterning different aspects of sound at multiple scales. We

return to patterns and TidalCycles in chapter 6, where we compare and contrast different approaches to time.

The capacity for algorithmic pattern to connect disciplines, including with the ancient history of heritage craft practices, allows the practice of live coding to be grounded and enriched. It also allows practitioners to connect their work with their everyday life experiences. In a review of live coding and algorave culture from a feminist technological perspective, Joanne Armitage quotes an interviewee speaking about code as "a way of working through their daily life, adding structures to it and providing functions for being. These lived patterns merge with their daydreams and expressions of color and geometry to form her live coded visuals."[41] In looking beyond the conventional grounding of computing practice in military, industrial, and commercial contexts to the far older and therefore more advanced ethnomathematical roots in pattern, it becomes much easier to see how live coding can develop as a cultural practice.

Machine Collaboration through Notation

Predictive coding is a term used in neuroscience for the way the brain filters out redundant noise (according to context) so that cognition can focus on the perceptual data that are relevant to the task at hand. However, with advances in the field of machine learning, we are getting to the point where a live coder is able to program in collaboration or conversation with a live coding system that has learned about their habits and style. Tanimoto has described levels of liveness in programming languages, where the sixth level represents a state where "rather than simply making tactical predictions, a system might be capable of successfully making strategic predictions."[42]

A number of projects have indeed explored what could be called *artificial live coding*, either as autonomous systems or assistants that suggest edits to human live coders. Interestingly, even relatively straightforward approaches to code generation turn out to be remarkably successful. Ixi lang's autocoder, perhaps the first practical example of artificial live coding that effectively manipulates code based on straightforward rewrite rules, is often used by performers who are the worse for wear or otherwise in search of inspiration.

Research into artificial live coding was prefigured by McLean's work on generating continuations based on Kurt Schwitters's sound poem *Ursonate* (1932),[43] for which he created a domain-specific language to represent rhythm that later formed the basis of the TidalCycles live coding environment.[44] Perhaps because TidalCycles has been designed to be easily readable by both humans and machines, it has since been used as the target representation for a number of artificial live coding systems. Shawn Lawson and Jeremy Stewart's Cibo agents have been developed over several iterations, version

two being trained on code recordings of prior TidalCycles performances, producing a three-dimensional latent space traversed by a recurrent neural network in order to generate a new performance.[45] Simon Hickinbotham and Susan Stepney applied evolutionary algorithms in a system for multiple autonomous agents to live code TidalCycles music together using the Extramuros network music system.[46]

The systems mentioned so far mainly operate only on the notational level, but the MIMIC project brings machine-learning and machine-listening techniques together in a system that encourages end users to create their own live coding languages. A strand of that project, called Sema, developed by Francisco Bernardo, Chris Kiefer, and Thor Magnusson,[47] invites people to create their own live coding languages specifically for machine learning in live coding practice. The MIMIC project workshops have invoked some of the early atmosphere of the live coding community where everyone worked with their own experimental notation and mixing live coding and artificial intelligence aspects also connects with Engelbart's early conception of augmented intelligence. We speculate further on the future of artificial live coding in chapter 8.

Live Notation in Performance

The function of notation in live coding can be seen as threefold: it is the syntactic structure read by the language interpreter that executes the program, it is the action or movement of the performer that is projected to the live audience, and it is the artistic (e.g., choreographic, musical, visual) output of the process that is notated and manipulated by the live coder. All of these require further unpacking, but let's start with the last point. In musical terms, the live coder is concurrently playing and composing. The computer (or any system of interpretation) is executing the music: the individual sonic events are not triggered by the musician. The written code serves as a trace of this conversation with the computer, but since the effects of this conversation are still sounding and transforming (rhythmic stuttering, melodic canons, shuffling, shifting patterns, and so on), the performer will need the code on the screen to consider what they have written and its relationship to the results.[48] The code is a representation of the sounding music process, and through reviewing it the performer plans the next steps, which might include adding new structures, changing running patterns, or deleting parts of what is happening. For this reason existing lines of code in live coding languages represent data structures, functions, objects, or agents that can be altered during performance without interruption in the execution of the artwork.

The relationship between code and results is, however, not straightforward—if it were, the live coder might be better off using conventional sequencer software designed

for working directly on the musical surface of notes. For example, when live coders compose together elements of different lengths, they create polyrhythmic interference patterns, bringing a result with features not present in the source elements. Similarly, if they are live coding behaviors of agents, the musical results are not directly described in the code but are an emergent property of how the agents interact. Conventionally, programming techniques are divided into either an imperative or declarative approach,[49] the first addressing the question of *how* the program should run and the latter addressing *what* the program should achieve. Live coders instead tend toward the question "What if?," where the notation is not used to describe a desired procedure or outcome but instead to simply take the next step in an exploration. Each such step is guided more by the coder's musical results of the previous step as they are perceived and less by an overall plan. The role of notation in live coding then is not to define, prescribe, record, or transcribe but to take a step into the darkness, into which the interpreter immediately throws light.

Recording Live Coding

Live coding is notational, digital, and discretely symbolic yet kicks against the assumptions of recording and reproduction. As we have seen, in many ways it is an example of oral culture, especially where live coders celebrate not saving their code. The transience of the code they write is an essential aspect of being in the moment, held in dramatic contrast to the life of the professional programmer who saves incremental versions, configurations, and releases. Nonetheless, all live coding gestures are via computer systems, which are generally deterministic. If a live coding performance is documented—for example, through a screen recording or recording of keystrokes—such a performance may in a sense be perfectly reproduced. However, one aspect applies here as to any improvised art, whether music, dance, or happenings: the documentation can never capture the unique spatiotemporal moment in which the performance takes place, which is dependent on the social context, historical time, and architectural space.

Live coding therefore treads an uncommon path between oral and written culture, improvisation versus the composed, and the spontaneous versus the arranged. Live coders often start with a blank page, but behind every written function is precomposed or pre-grammed code, often encapsulating a great deal of music techniques, such as syncopations and other transformations. Some live coders operate like jazz musicians: they practice "licks" that are applied in the live context, composing pieces in real time that have to some extent been practiced. Live coding is written, like music notated in staff notation, but it originates as a tradition in which composition happens in real

time, and the results are often abandoned: this music is about the process, the experience in-the-making, and not about the destination.

Increasingly though, Save buttons are disappearing from text editors and word processors because every keystroke triggers a save into the complete history of a document. Experimental live coding systems, such as Troop and FeedForward,[50] are adopting this model, too, following Sang Won Lee's work on *live writing*, in which every key press is saved with a time stamp and able to be recalled and replayed.[51] This signals a shift away from ephemeral live coding that is made for the moment to plentiful live coding, where every action is shared immediately to the creative commons. This is an alternative stand against commercialization; rather than deciding not to save code so there is nothing to commercialize, everything is saved and shared so there is no scarcity to exploit. This brings to mind Mark Fisher calling for collective wealth: "Real wealth is the collective capacity to produce, care and enjoy. . . . This is Red Plenty. . . . Everything for everyone. All of us first."[52]

Between Speech and Writing

Although musical notation systems can be found all over the world, emerging along with writing, musical composition using notation is a relatively recent phenomenon (about one thousand years old). Musicians in many cultures, from the dance music of the Ewe people of Ghana to Hindustani classical music, still rarely show interest in notating their practice. These advanced practices have developed through oral transmission (standing for any form of communication, whether sung or instrumental, other than writing), and although the music *could* be transcribed, its practitioners rarely have cause to do so. In contrast, Western classical music lives through its notation, with music composed, shared, played from, analyzed, and theorized in printed form. This relates to the advent of the printing press in the fifteenth century and the distribution of sheet music, offering new ways of generating income for musicians and composers. These different practices in music culture appear to set a dichotomy between music primarily shared through speech and through play and that shared through notation. Most music cultures tend to have a mix of both but lean toward one or the other.

Live coding sits uncomfortably on the division between oral and written culture. It questions how speech and writing are transformed through their relation to computation and coding practices.[53] On one hand, an archetypal live coding performance focuses on notation to an extreme—everything happens via text. Furthermore, that text is not a mnemonic but a *complete* description in that it is a formal language interpreted unambiguously by a computer. Indeed, rather than a description it is a *prescription* because it

does not describe music but brings it into being. While a human instrumentalist brings their own creative interpretation to playing notated music, a computer interpreter has no such role, beyond perhaps pseudo-random number generation as dictated by the notation. On the other hand, the manner in which live code is articulated is ephemeral. Live code is only notated in order to be changed and, in many cases, deleted. So in a sense, the fleeting, momentary nature of a live coder's notation is closer to speech than text.

Live coders therefore find themselves caught between two worlds. They work with a written notation, distanced from what is notated. They don't directly move to make gestures; they *write about* making gestures. But they still continually manipulate that notation while it is being interpreted, changing lines of code by adding new features and deleting others (we could call these *meta gestures*), and as a result, live coded notation cannot later be printed out and shared.[54] The art is not in the written notation itself but in the *activity* of notation, as performance over time. The programming is part of the program.

We can only conclude that live coding is neither a fixed text nor dynamic speech. However, we will get nowhere by focusing only on differences between practices. We therefore look to understand live coding in terms of how it breaks apart conventional categories and is able to reframe creative practice in terms of live manipulation of symbols, and conversely, we look to reframe live coding as fully embodied interaction through code. That live coders oscillate between literary and machinic modes and practices in live performance—somewhat analogous to speech—leads us to speculate on whether their live writing retains some of the unruly qualities of speaking out (in public). Or perhaps live writing is another way to highlight the unfree conditions under which all communication operates and offers one way to examine its effects and conditions of operation. To write differently, developing new performative modes of expression becomes all the more urgent if we understand human subjects to be constituted in language, as well as able to transform that language and break out of processes of subjectivation.

Paper Rhythms: The Map Is Not the Territory

Previously, we made the claim that live coding notation is in a sense a *complete* prescription for music, as opposed to a mnemonic form. This, however, does not mean that there is a clear relationship between a notation and the real-world experience conjured up by its interpretation. The way we perceive a piece of music may be structured very differently from the way we notate it. This could be a disturbing thought for the live coder, caught between perception and notation.

In order to understand the relationship between notation and what it notates, we can turn to music analysis. Musicologist Kofi Agawu signals a warning for those trying to understand music only through analysis of notation, calling those *paper rhythms*.[55] For example, in an analysis of notated rhythmic timelines such as the African standard bell pattern, it is common to depict them as circular and, as a result, treat "rotated" time lines as equivalent in the same way that octaves are equivalent. According to Agawu, the mistake here is to assume that the *correct* time signature to assign to a rhythm is the one that is easiest to notate, and claims that follow from this mistake are easily refuted by talking to musicians who play the music, by speaking the language that the rhythms relate to, or simply by dancing to a rhythm to feel the true underlying pulse and meter. By looking for musical relationships in notation, we confuse notated paper rhythms with rhythms as they are experienced in the music, grounded in physical and not purely symbolic relations.

On the face of it, Agawu's criticism could be disturbing for live coders because he criticizes music analysis that equates rhythmic representations that look good on paper with those that work well in practice, when these are two very different domains and experiences. But that is what live coding musicians do—they work with paper rhythms as live, musical practice.[56]

Consider the following pattern in the TidalCycles live coding environment:

```
every 3 ("t" <~) $ struct "1(7,12,3)" $ sound "drum"
```

This plays the twelve-beat African standard bell pattern (notated as the Euclidean pattern (7,12,3)[57]) and "rotates" it by one-quarter of a cycle every third repetition. But with his authority as a musicologist of both Western music theory and the music of the Northern Ewe, Agawu would argue that this is only a rotation *on paper* and that what we hear and feel is not a rotation but an entirely new pattern, with very different properties. This relates to the concept of *symmetry breaking*, in which you break symmetry by applying a force to it, which then potentially causes a whole new symmetry to spontaneously form.[58]

Anthropologist Tim Ingold's view on notation, referring to Alfred Korzybski's famous statement "The map is not the territory,"[59] helps us understand this problem from the perspective of and mapmaking navigation:

> Do we say that a notation allows a musician to create a piece, as it was intended by a composer, "from the coals?" In the sense that the notation contains all the instructions to produce this piece? Or is the notation such that you can't understand it, unless you know how to play it already? Many notations are of that latter kind—they are maps which you can only read if you already know the territory.[60]

If music notation is a map for those who already know the territory, then perhaps live coders are live mapmakers of *unknown* territory. That is, a live coder is making a map-as-code, from which the computer generates a result, which the live coder then experiences as territory-as-sound. Having had this experience of the sound *territory*, they are able to read the code *map* they have written for the first time. From then on, each adjustment to the map in turn adjusts the territory, allowing a new reading of the map and a new adjustment and so on.

This generative relationship between a notation and what is notated is what drives creativity in the live coding field. The perceptual gap between notational map and generated territory allows the live coder to reach beyond their imagination, working with notation not to efficiently realize a ready-formed idea but to *follow* an idea to see where it takes them.[61] Rotating or reversing a rhythm might be a simple transformation on paper that when actually played out becomes a transformation of a complex whole, with the phenomenon of syncopation resulting in a rhythm with a completely different feel. In enacting such transformations, the live coder is in a perpetual state of anticipation, with each edit a hypothesis that might be confounded but nonetheless informs the next edit, allowing the live coder to guide a performance into the unknown.

What we then arrive at is the idea of a live coder who is simultaneously both theorist and practitioner. They are not beholden to established music or choreographic theory (except perhaps theory embedded in the live coding language they use) but instead continually develop, test, and share new theories. Each edit is a prescription for a method that is part of a wider holistic process that encompasses the human perception of rhythmic, kinetic, visual, tonal, and/or timbral relationships. As Klee put it:

> Already at the very beginning of the productive act, shortly after the initial motion to create, occurs the first counter motion, the initial movement of receptivity. This means: the creator controls whether what he [*sic*] has produced so far is good. The work as human action (genesis) is productive as well as receptive. It is *continuity*.[62]

Holistic Prescription

The distinction between holistic and prescriptive forces in live coding can be further clarified from metallurgist and research physicist Ursula Franklin's conception of the "real world of technology."[63] For Franklin, a holistic technology is exemplified by the craft of an artisan who is in control of their own creation process and where every piece they make is therefore unique. Prescriptive technology, on the other hand, concerns a creation process that is broken down into tasks, each well defined and reproducible. Here Franklin explicitly considers *technology* as *practice* and therefore discusses

technology not in terms of *what* is done but *how* it is done. Holistic technologies follow a growth model, in which the artisan makes decisions as they go, crucially including the decision of when to stop before the work becomes *over*grown. By contrast, prescriptive technologies follow a production model in order to support division of labor and mass production, whereby makers do not assert control over what they are doing and instead are fully compliant in following their assigned subtask.

This distinction again confronts us with an apparent conflict at the heart of live coding. On one hand the prescriptive "breaking down of a process into well-defined tasks" is what all computer programmers do through the decomposition of a problem into more easily solvable tasks. On the other hand, the liveness in live coding pushes a holistic role for notation, in which the coder is a craftsperson working with existing programming environments at different layers of abstraction and responding to their code as the material of creative constraints. The unambiguity of code, and its use in prescribing a deterministic process, can blind us to the wider, fundamentally human creative process at play. What a live coder unambiguously prescribes is action, but where that action creates results beyond their imagination, grounded in the creative responses of our perception, we arrive at a holistic process open to unintended consequences and surprise.

An alternative view of the live coder is of an artist who is "at one" with their tools, to the extent that they know the result of every action in advance and anticipate everything their software does. This is a live coder as a virtuoso, channeling art from the Platonic world of ideas into an imperfect world. Notation is then a matter of efficiency of expression, of getting what is in the coder's mind into the world in the most expedient way possible. One problem with this view is that if a live coder has full control over what their notation produces, then in terms of its creative influence, notation is neutralized and sidelined. Rather than allowing for an exploration of language with all its generative capabilities, live coding is then reduced to getting a result already predefined by the coder. In this case, perhaps, they would be better with a system like a digital audio workstation (DAW) or a sequencer, designed to allow the direct manipulation of every note. In practice, however, live coding exists between these extremes. The language is always in dialogue with the live coder, its affordances and constraints presenting themselves through live action. A live coder might well arrive at a performance with some particular ideas to express while leaving space for new ideas to emerge.

In some cases, the live coder might be the only person onstage, but we should not forget that there are always other people involved in setting the scene. As live coding has developed, naturally most now perform using languages created by people other than themselves. These live coding languages might be Turing-equivalent and therefore universal, in that any process can be described using them, but nonetheless

they are epistemic tools that offer particular creative constraints and affordances put there by the language author.[64] As Roland Barthes wrote in his essay "The Death of the Author,"[65] it is language that speaks, language that performs. In a sense, then, the live-programming environment prescribes a particular world of possibilities, and hopefully a rich one, allowing the live coder to creatively explore and push against those constraints as well as define their own.

Space in Notation

Another dichotomy that is core to the design of live coding notations is between *visual* and *textual* programming. This distinction *seems* clear at first, wherein visual programming languages build upon a visual layout and graphic elements, and textual languages are based on arrays of discrete symbols. When we look more closely at textual and visual languages, however, we see that this distinction stands on shaky ground.

Figure 4.2 shows the Sonic Pi live coding environment using one of its default themes. The performer has placed a webcam image underneath for this online livestream, but otherwise this is a standard presentation of the software. Sonic Pi is very much a textual live coding environment (based on the industrial Ruby programming language), but in this screenshot we see that its text both lives within and is supported by the visual

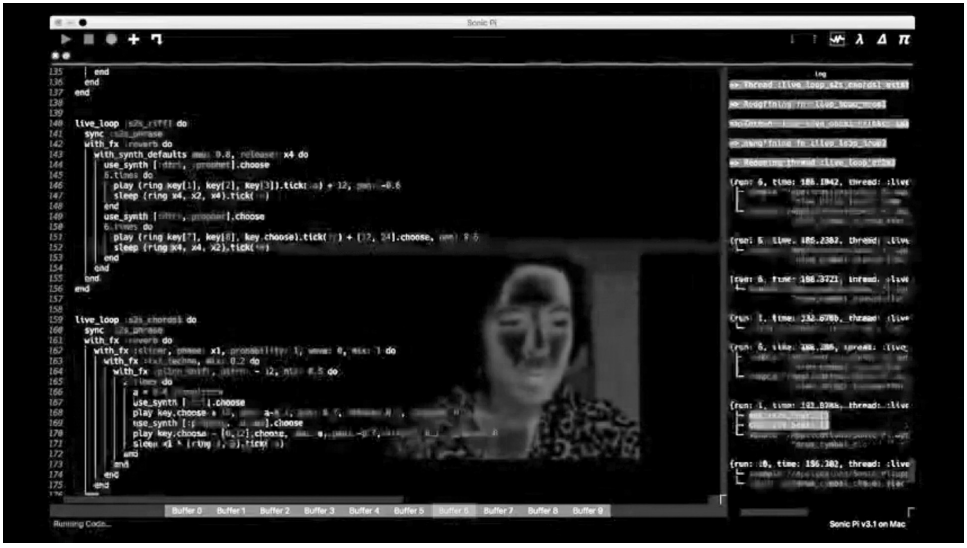


Figure 4.2
Melody Loveless performing as part of the New York DigiAna stream, September 12, 2020.

domain. The text is arranged within a two-dimensional grid, and furthermore, text is visuospatially placed *within* text through indentation, with alignment accentuated by vertical lines, creating a visual representation of embeddedness. Color visualizes syntactic elements. Adjacency visualizes connectedness. Time proceeds downward and loops within horizontal levels of indentation. These are all visual relationships.

The visuality of text is clearer still in figure 4.3, which shows the Orca live coding environment. In conventional programming languages, control flow proceeds downward, but in Orca, control flows can travel up, down, left, or right, triggering effects (including change of direction) indicated by single-letter (ideographic) commands. Control flow is itself visualized as highlights moving around the text and may self-modify instructions as it travels around. Orca has a great deal in common with the older Befunge programming language; the only real difference being that Befunge was created as an "esoteric" language not intended for "serious" use, whereas Orca is a practical, and increasingly popular, way to make music. Each letter is treated as an ideographic symbol rather than being composed into words, but it is still undeniable that this is a text-based language, which includes both visual syntax and semantics.

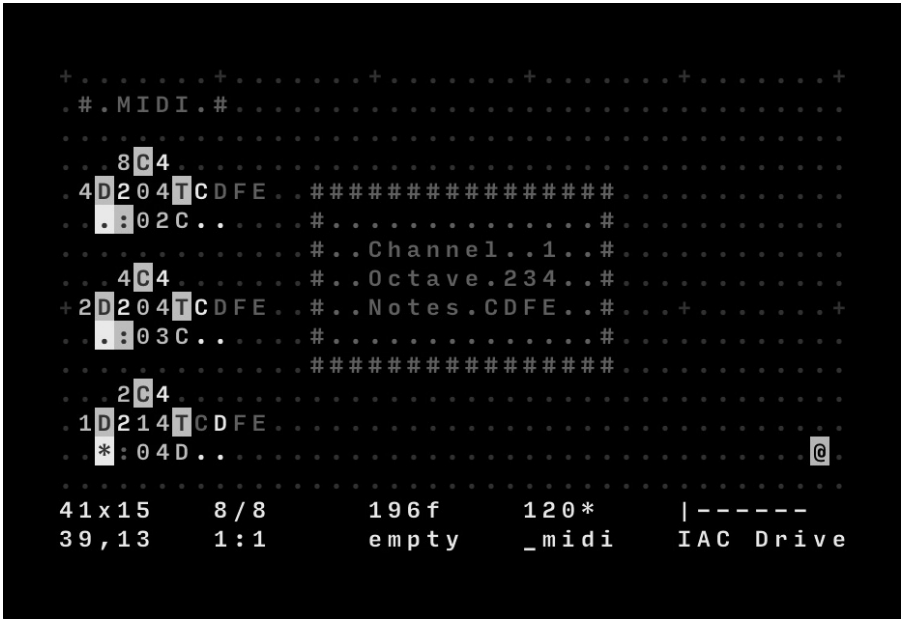


Figure 4.3
Orca live coding environment.
Source: Screenshot by Alex McLean, generated using Orca software by Devine Lu Linvega.

If we look at familiar visual languages such as Scratch, Pure Data, and Max, we find that all involve text but have alternative means to visualize syntactic connectedness. In earlier generations of textual programming languages, local syntactic connectedness is determined by the sequence of symbols, just as with the English-language text you are reading. In Scratch, the local sequence is visualized as jigsaw-like blocks, while with Pure Data and Max connectedness can be indicated over greater distances by a line drawn between text labels, each contained in a rectangle. So, in a very real sense, visual programming is textual, and text-based programming is visual. The specialist research communities in visual programming languages and diagrams continue to explore new forms of these text/graphic hybrids, with theoreticians further problematizing any naive graphic/linguistic distinction.[66]

Several live coding environments use the ambiguity and interdependence between textual and visual representation to create hybrid interfaces, combining visualization and a graphical user interface (GUI) with text. Some of these environments take a game-like form, such as Griffiths' work based on his live coding game engine Fluxus.[67] Others keep the appearance of a standard text editor but are augmented with GUI elements. The web browser–based Gibber live coding environment, created by Charlie Roberts, has been a particularly active and influential platform for experimentation in live code annotations and visualizations.[68] Graphic elements allow the end user to associate the sounds they hear with the text that caused it, with active elements highlighted and continuous signals visualized. Furthermore, individual values can be interacted with and adjusted with a mouse. As with Orca, these features are strikingly unusual yet also highly practical in bringing live text into a whole audiovisual experience.

This visualization of control flow in the code that notates it amounts to a conflation of space and time in the text editor. Earlier environments have explored this too. Magnusson's ixi lang, for example, has turned code lines into notation of rhythmic time, where symbols represent musical events, and spaces between the symbols then signify musical "rests."[69] Ixi lang has functions that operate on the code as data, where the code invokes a process for editing itself. We return to the representation of time in live coding environments in chapter 6.

Notation and Authenticity

With notation comes the possibility of assigned authorship and, by extension, authenticity. How does this apply to live coding practices? Nelson Goodman's 1976 treatise *Languages of Art* distinguished two fundamental categories of artistic work.[70] An *autographic* work, like a written signature, retains traces from the body that made it. When

viewing an autographic work such as an oil painting or sculpture, we see substance that has been shaped by the hands of the artist. There is a chain of material action connecting the viewer to the original performance—an act of expressive making. Although it is possible to copy such a work, reproducing that original performance with greater or lesser fidelity, we feel obliged to distinguish between such activities and the original work. One might attempt to pass off a copy as if it were the original, but this would be a different kind of expressive performance—an act of live forgery, not live creativity. Goodman's second category is that of *allographic* works. Unlike an autographic work, an allographic work can be reproduced without changing its nature. If I own a printed copy of the novel *Sense and Sensibility* and I see another copy in a shop, that copy is just as much the work of Jane Austen as my own. Making another copy is not a forgery of a novel or a score. In the case of allographic works, there is no fundamental difference between copies, and the essence of the work is the information that is encoded in every one of the copies. Reproduction of code is not forgery—it is not even live, in the sense that it can be achieved wholly mechanically.

Goodman labels the fine art practice of painting as autographic and the process of producing a musical score as allographic, but as a technique live coding can be applied both to making paintings and making music. So on which side of this distinction does live coding sit as a field, and what could it mean to make an "authentic" live coding performance? When Goodman writes about music, he has an orchestral performance in mind, with musicians reading from a score. It is less clear where music improvisation would fall, but the comparison between live coding and orchestral performance is still interesting since Western classical music is "two stage" in that there is the notation and its execution. Live coding brings notation and execution together, taking place at the same time in response to one another.

When a live coder works as a live improviser, sharing their screen, this would seem to be autographic—the body can be seen to make the text in the process of generating live performance outcomes. Perhaps the authenticity is then in the nonrepeatability of the performance. Has the live coder rigidly practiced every edit and keystroke in order to reproduce a work? If we view live coding as an allographic practice, then this is an authentic reproduction of the piece, but if we view it as autographic, then this is a "fake" improvisation! In reality, these are two extremes at opposite ends of a spectrum, where a given performer is likely to move between preprepared and more improvised sections at will. In the same live coding event, we might see one performer recalling and executing preprepared code, the next writing everything from scratch, designed in and for the moment, and a third taking a hybrid approach. We can say then that the live coding community supports both autographic or allographic approaches, celebrating

the former for its open sharing of creativity and risk and the latter for its slick control of a well-prepared composition.

In practice, the notions of allographic and autographic works are as interwoven as the notions of analog and digital, and the problems of seeing them as separate have been laid bare by the rise of digital media.[71] We call an MP3 file "digital" because it represents sound using discrete mathematics, stored as numbers. However, ultimately, the MP3 file represents analog movements of air pressure; the file is digital, but the outcome is very much analog. This is a layering of analog movement represented in a digital file, which in turn is held within the analog form of a silicon circuit. This layering allows an autographic work to be mass produced perfectly without the requirement of an original. The distinction between allographic versus autographic is similarly difficult to make in oral culture. The discrete, digital elements in a song—the words and notes—are handed from person to person as living memory, changing with use. The words and notes are specific, able to be transcribed and recorded as digital media, but an exact copy has less value than the next performance, which carries the autographic signature of the performer on to the next one.

John Cage alludes to the mass production of algorithmic music as an escape from the questions of authorship that underlie the distinction between allographic and autographic works: "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. Subroutines are altered by a single punch. We're getting music made by man himself: not just one man."[72] Both chords and subroutines are digital in nature in that both consist of discrete, countable elements. The discrete symbols used in live coding are similarly caught up in a life of continuous change, the digital mixing with the analog. A live coder works in a digital world of formal language, but that code ultimately describes continuous movements, which the coder experiences both as discrete events and continuous fluctuations.[73] Similarly, we find it hard to describe a printed novel as being a digital artwork; it may exist as a list of discrete alphabetical symbols, but when read, the experience is neither analog nor digital but an amalgamation of both.

Digital media has problematized autographic artwork, reflected in the ongoing debates over copyright ownership and commercial distribution. However, the nature of allographic works is even more significantly modified by digital media. The classical allographic work was a script or score, in which the encoded work itself is distinct and definitive, but the code specifies performances that may vary from each other. However, any digital work can be regarded as a score, specifying the action to be taken by a computer. When rendered via digital processes, the coded specifications of an

artwork seem to be not so much performance scores but computer programs. A computer program, like a script or score, specifies a sequence of actions to be rendered in "performance" when the program is executed. Computer programs, like scripts and scores, are written with specialized notational conventions designed to be unambiguous, compact, and expressive of the intended performance. Their purpose is to describe events that should take place in the future.

However, digital scores are also able to capture and preserve live performance. The distinction between specifying performance and recording has often been problematic in the performing arts. Laban dance notation is used to summarize, abstract, encode, and transcribe body movements. Despite that capability, when a dancer needs to learn a work, they more often learn it either by instruction from a choreographer, by transmission from other dancers who are already familiar with the piece, or from recorded video. Likewise, choreographers seldom construct traditional notation as a prescriptive score, preferring to demonstrate their intentions directly with their own body or that of a dancer. Nevertheless, the notion of notation (with all of its problems and potentialities) remains a focus of interest for many in the field of dance.[74] Just as a musical performance of a notated score in the Western tradition is an interpretation, so these notated recordings can be used to interpret and represent performing actions.

We Are All Live Coders

Every element viewed on a computer screen is a notated representation, either encoding and recording an action that the user has taken or specifying an action that the machine should take in future. These actions might include playing sounds, sending messages over a network, or even controlling machinery, but more than anything, the computer operates to modify itself. The internal state of the machine memory is an immense catalog of notation—recorded actions and planned behaviors. The notation of the screen is only one view, a window into this world of internal representations. Even more than the printing press, the media industry, or bureaucratic structures of government, the computer is a representation machine. Operating a computer is a constant manipulation of notational representations, and every computer user has the opportunity to think of their actions in terms of mathematical abstractions, structuring and adjusting the state of digital memory.

The modern GUI of icons and menus blurs the boundaries between data and programming language. Originally designed at Xerox PARC as part of the Smalltalk programming language, emerging from the Kiddikomp proposal for children to make their own art and music, the GUI has evolved into a universal diagrammatic notation—a visual

language combining a lexicon of pictorial words with a syntax of lines, shades, borders, and typographic design cues. Every computer user constantly learns to speak new notational languages, each of them offering a semiotically mediated conversation with an application designer who has offered possible behavioral functions via that user interface. To some extent, every computer user is a programmer—manipulating abstractions, combining software components, and modifying the future behavior of the computer. We read, interpret, consider, and redraw. When we use these machines as reflective abstract tools, perhaps we are all live coders, all the time.

While traditional art forms have conventionalized and internalized their notations, to the extent that a musician may experience a direct connection from the page to the keys of an instrument, these digital notations are in a constant flux of innovation. In a conventional musical score, the distinction between, say, the key signature (nonnegotiable) and performance markings (open to negotiation/interpretation by performers) is conventionalized and seldom highlighted as problematic. But when a new notation is invented, the correspondence between the graphic marks and the intended interpretation must be negotiated between the software designer, the system user, and the changing state of the application itself. Linguistic analyses such as Jacques Bertin's *Semiologie Graphique* and Yuri Engelhardt's "The Language of Graphics" offer a variety of modes of correspondence,[75] potentially determined by geometric constraints, symbolic conventions, or novel metaphorical interpretation.

The algorithmic interpretation of formal notation elements is often supplemented by *secondary notation*—remarks intended for a human reader that escape the constraints of the formal computational scheme. In a performance context, these may be notes for the performer, a side channel of commentary to the audience, or a resource for gestural control. In all of these cases, we might reflect on the questions raised by Goodman. Is the live coder allographically specifying a distinct machine state or autographically shaping artistic material? Is the computer recording, duplicating, or interpreting? Are the diagrammatic elements symbols selected from a discrete vocabulary or infinitely expressive visual markings?

These distinctions become increasingly unstable in virtual reality and in tangible and embodied interaction contexts. Much of the above analysis remains applicable. Digital control representations are still diagrammatic notations, even when disguised to resemble imaginary objects or environments. The correspondence from virtual appearance to expected behavior may become more surprising or baroque but will continue to be constrained by basic principles of geometry and visual perception. The relationship between recorded data and specified behavior becomes increasingly hard to disentangle when machine-learning and program-synthesis techniques derive programs from

observations. Increasingly, digital instruments resemble live coding languages, with the sound-processing variants of the loop pedal and harmonizer offering a resource for the algorithmic automation and augmentation of every action.

Dynamics of Machine Notation

Sheet music stays put on the page. The content of the page has been determined by the composer, editors, and engravers, and their work has ended when the music starts. Exceptions, such as a composer at a musical premiere crawling between orchestra desks to make last-minute changes to the copied parts, are memorable for their rarity. Even more open works—graphic scores, or textual instruction pieces for avant-garde happenings—take the printed page as a boundary object that allographically structures the relationship between composition and performance.

But when we place the notation on a screen, it becomes potentially mutable, permeable, ephemeral. As we've seen, live coding environments such as ixi lang, feedback.pl, betablocker, and Orca change themselves—manipulating the source code of the program itself like any other data. The familiar diagrammatic notations of the GUI desktop or mobile phone screen are conventionally static, but even this constraint is adopted purely for the convenience and comfort of the user. In many systems, the visual notation results from the underlying data, rather than determining them.

Heritage performing technologies, such as the violin or tabla, carry out a kind of conversation, or perhaps mutual oscillation, with the performer, offering tactile feedback through the hands and body. The performer constantly attends to these signals, shaping the sound through a collaborative accommodation of expressive intentions to mechanical constraints. Although *hybrid live coding* is becoming an active field of study,[76] we have yet to discover the full potential of such conversations for the live coding performer. A dynamic notation can respond to the instructions of the author. Trivially, this might be to express a constraint, such as a syntax error or runtime bug. Modern software-engineering tools offer a variety of more interventionist opportunities. Improvising programmers make extensive use of autocomplete functionality, in which the program editor anticipates the likely intention of the programmer and suggests the next fragment of code.

When a computer interprets source code, it becomes another reader, in addition to the live coder and audience. Whereas a score often presumes expressive interpretation by the performer, the live interpretation of the code to produce a dynamic visual or aural performance is carried out by the computer that executes the program. A central point is the sharing of agency between the person writing code and the machine executing it. Live coders often introduce variation through the use of random or stochastic

functions that at the level of notes and phrases reduce predictability. This might involve varying a continuous parameter according to a statistical distribution around a mean or shuffling the order of sounds and events. Over longer time frames, such decisions made using random number generators quickly lose their impact because they are arbitrary and thus predictable in their unpredictability. However, as explored earlier in the section on algorithmic pattern, unanticipated patterns and symmetries arise naturally from the combination or transformation of elements. Such patterns may unfold over multiple scales, giving longer form structure to performance.

Pattern Language of Lively Architecture

We can reflect on the attributes of notational space, and the experiences of those inhabiting it, through analogy to architectural theory. The work of architect Christopher Alexander has been influential in computer science, particularly for his work on pattern language.[77] Alexander's work focuses solely on the physical architecture of buildings, but it has been imported into software engineering in the form of a well-known set of object-oriented design techniques.

Of course, something has been lost in translation. For Alexander, dwellings and cities born from contemporary architecture lack life because they are now designed by distant architects, not designed and continuously modified by the people who live in them. Alexander's pattern language aims to provide a set of transformations for use in architectural design to solve problems in buildings and make them more lively and nurturing. Here his notion of lively architecture seems closely analogous to the motivation for live code. Indeed, as Alexander noted in a keynote lecture to computer scientists,[78] while the problem-solving aspect of his pattern language has been successfully brought to computer programming, the moral dimension has not. While Alexander was optimistic in his 1996 lecture that computer science could save the world where architects have failed, from our perspective in 2022 optimism for this view is in short supply—it seems we are still waiting for software engineers to take on the moral capacity to generate the coherent, living wholes that Alexander advocates.

Nonetheless, a direct reading of Alexander's work helps us reflect on the attributes of code as notational space and the experiences of those inhabiting it.[79] Alan Blackwell and Sally Fincher offer a pattern language of notational systems as a systematic description similar to work in notational spaces. These patterns of user experience are derived from Green's "cognitive dimensions of notations,"[80] which have previously been used to analyze usability of sequencers,[81] historical manuscript typesetting,[82] DAW,[83] and novel graphic scores.[84]

Working within a virtual notational space is unavoidably dynamic, resulting in patterns of experience that might be characteristic of reading, transcription, modification, or exploratory improvisation. As we construct and enhance new notations, a design pattern language of this kind draws attention to the potential for new modes of interaction, as well as opportunities to recognize the underlying cause of frustrations or loss of fluidity. Unlike conventional musical instruments, whose limitations and constraints are familiar to the expert performer, every innovation in a live coding language navigates an abstract space of design trade-offs.[85]

Social Relations Expressed through Notation

The role of notational systems in culture is now under scrutiny given our increased dependency on scripts, scores, and algorithms, which shape our decisions and behavior. Trying to understand the underlying logic behind them becomes urgent for those who are able to read and write them. This is especially important when algorithms work on big data at scales that are hard to conceptualize, but there is a longer connection to politics that we briefly touch upon here and elaborate on further in chapter 7.

A frequent reference for the role of scores in culture is the music theory of Theodor Adorno and his scathing views on jazz and "popular" music as standardized commercial form. In his 1938 essay "Über den Fetischcharakter in der Musik und die Regression des Hörens" (On the Fetish Character in Music and the Regression of Listening), he considers music to be a by-product of the musical score, which represents a purer form associated with production.[86] Once the score is performed, the listener becomes a consumer of the commodity form of music—the commodity fetishism of music, in other words.

In contrast to Adorno's view, live coding traverses some of these relations between notational practices and performance. Of course, there is little new in the romantic idea that live music somehow breaks out of commodification—with the various slogans to "keep it live"—but live coding is both notational practice and performance at the same time, allowing for new critical potential. The production processes are open to view as changes to the code are made public as part of the performance itself.

Related examples might help to establish this point, especially when notation is considered to be a social project. There are numerous historical references to alternative notation systems and improvisation techniques, as the previous section outlined, but the experiments of Cornelius Cardew and the Scratch Orchestra emphasize the political potential of notation. Simon Yuill, in his essay "All Problems of Notation Will Be Solved by the Masses,"[87] makes the further connection between experimental

notations and free software development, as well as the practice of live coding as an instantiation of making source code available and modifiable in real time. Each of these examples attempts to break out of standardized commodity forms of software development and electronic music performance, respectively.

The Scratch Orchestra emerged out of various critical energies of the late 1960s and managed to develop a collective form for the sharing of resources, self-organization, and peer critique. The orchestra was open to all, regardless of musical training or ability, under the principles of free improvisation. Notes, or "scratches" as they were called, were performed and developed into larger collage forms, similar to the ways in which source code is shared and distributed, open to further modification, and performed under "copyleft" principles. Their works played with organizational forms and hierarchies, as in the following instruction piece cited by Yuill: "Each person entering the performance space receives a number in order. Anyone can give an order (imperatively obeyed) to a higher number and must obey orders given him by a lower number." This score resembles the comparisons in a *bubblesort* algorithm and exemplifies the command and control structures of computational systems and the parallels between

Figure 4.4
"Keep Live Coding Live" sticker.
Source: Design by Alex McLean.

technical and social systems. Cardew attacked the conservatism of musical notation and announced that "all problems of notation will be solved by the masses."[88]

This comes back to the rejection of end product or commodity form in performance. The political philosopher Paolo Virno, also cited by Yuill, develops this in relation to the idea of "virtuosity"—reworked from Aristotle via Arendt and Marx—to indicate the "special capabilities of a performing artist."[89] For Virno, what characterizes the "work of the performing artist" is that their "actions have no extrinsic goal. They don't create a lasting product since they aim only at their own occurrence. They don't create new objects, but rather a contingent and singular event. . . . The purpose of their activity coincides entirely with its own execution."[90] Virno is drawing upon Hannah Arendt's observation that the performing arts have a strong affinity to politics and that both operate in real time and exhibit their own sense of purpose embedded in their form. The problem is that the score has been appropriated to particular ends, whereas it should remain without end, as "virtuosity without a script, or rather, based on the premise of a script that coincides with pure and simple dynamics, with pure and simple potential."[91] Virno's discussion on virtuosity also offers a different slant on the relation between live performance and its reproduction. It is this lack of an end product—or at least one that is indistinguishable from the performance itself—that enables performing arts to be conceived of as a species of political action. Perhaps one should simply conclude that the script, score, and code might be held open to change and modification at all times and in the public realm. The slogan to "Keep music live" (figure 4.4) takes on an urgency that might be applied to further notational forms, and live coding seems to offer radical potential in this regard. That live coding is notation and execution at the same time provides this potential.

The MIT Press would like to thank the anonymous peer reviewers who provided comments on drafts of this book. The generous work of academic experts is essential for establishing the authority and quality of our publications. We acknowledge with gratitude the contributions of these otherwise uncredited readers.

This book was set in Stone Serif and Stone Sans by Westchester Publishing Services.

Library of Congress Cataloging-in-Publication Data

Names: Blackwell, Alan F., author. | Cocker, Emma, author. | Cox, Geoff, author. | McLean, Alex, 1975– author. | Magnusson, Thor, author.
Title: Live coding : a user's manual / Alan F. Blackwell, Emma Cocker, Geoff Cox, Alex McLean, and Thor Magnusson.
Description: Cambridge, Massachusetts : The MIT Press, [2022] | Series: Software studies | Includes bibliographical references and index.
Identifiers: LCCN 2022008717 (print) | LCCN 2022008718 (ebook) | ISBN 9780262544818 (paperback) | ISBN 9780262372626 (epub) | ISBN 9780262372633 (pdf)
Subjects: LCSH: Computer programming—Philosophy. | Agile software development. | Creation (Literary, artistic, etc.) | Algorithms—Psychological aspects.
Classification: LCC QA76.6 .B5794 2022 (print) | LCC QA76.6 (ebook) | DDC 005.1301—dc23/eng/20220527
LC record available at https://lccn.loc.gov/2022008717
LC ebook record available at https://lccn.loc.gov/2022008718